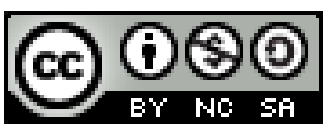
Introduction to OS/2 Warp Programming

Course Code: OS290
Version 2.9
Date: 1999-April-22

License and Contributors

- This course material is released under the Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License
  - http://creativecommons.org/licenses/by-nc-sa/3.0/
- The original author of this course was Les Bell and Associates Pty Ltd on 1997.
- The content was released by Les Bell and Associates Pty Ltd. under CC license on January of 2012.
- Martín Itúrbide from OS2World.com transformed the content to a newer format from Lotus Freelance and Word for OS/2 on 2012.



About the Course

- Course Timetable
  - Coffee breaks, lunch
- Phone messages, toilets
- You have copies of the slides used in the course
- Plus additional notes dealing with some topics in more detail
- Please feel free to ask questions at any time
- Each topic is followed by a lab session in which you will be asked to insert code in locations marked by comments
- Some labs are simple programs designed to avoid distracting complex code, others are moderately complex
- Everything discussed here is confidential

Lab Exercises

- Some labs are simple programs, designed to avoid extraneous detail
- Some are more complex, like real OS/2 programs
- Some involve design decisions
- To install, A:INSTALL C:
- Creates a C:\OS290 subdirectory
- Search the files for *LAB* to find missing lines or sections

Course Overview

- 286/386 Protected Modes and Memory Models
- Introduction to Tools
- Presentation Manager Programming
  - Anatomy of a PM Program
  - Window Parentage and Ownership
  - Window Classes
  - Menus and Window Controls
- Base Operating System
  - Memory Management
  - Dynamic Link Libraries
  - Processes, Threads and Priorities
- Advanced PM Programming
  - Window Words
  - Object Windows
  - Dialog Windows

Agenda

- Day 1
  - Session 1 – Introduction to Tools
  - Session 2 – Introduction to PM
  - Session 3 – Lab Exercise 1
  - Session 4 – Windows Parentage and Ownership
- Day 2
  - Session 1 – Window Controls
  - Session 2 – Lab Exercise 2 – Menus and Messages
  - Session 3 – Memory Management
  - Session 3 – Lab Exercise 4 – Memory Management
  - Session 4 – Dynamic Link Libraries
  - Session 4 – Lab Exercise 5 – Dynamic Link Libraries

Agenda

- Day 3
  - Session 1 – Threads, IPC and File I/O
  - Session 2 – Lab Exercise 6 - Threads
  - Session 3 - Workshop
  - Session 4 – Filesystems % EA's
  - Session 4 – Lab Exercise 8 – Directory Listing
- Day 4
  - Session 1 – Window Words, Subclassing, Dialogs
  - Session 2 – Lab Exercise 9 – Multiple Windows and Instance Data
  - Session 3 – Lab Exercise 9 continues
  - Session 4 – Standard Dialogs and INI files
- Day 5
  - Session 1 – Graphics Programming Interfase
  - Session 2 - Workshop
  - Session 3 – SOM and WPS
  - Session 4 – It's Friday…

Day 1 – Session 1

Introduction to Tools

Introduction to Tools

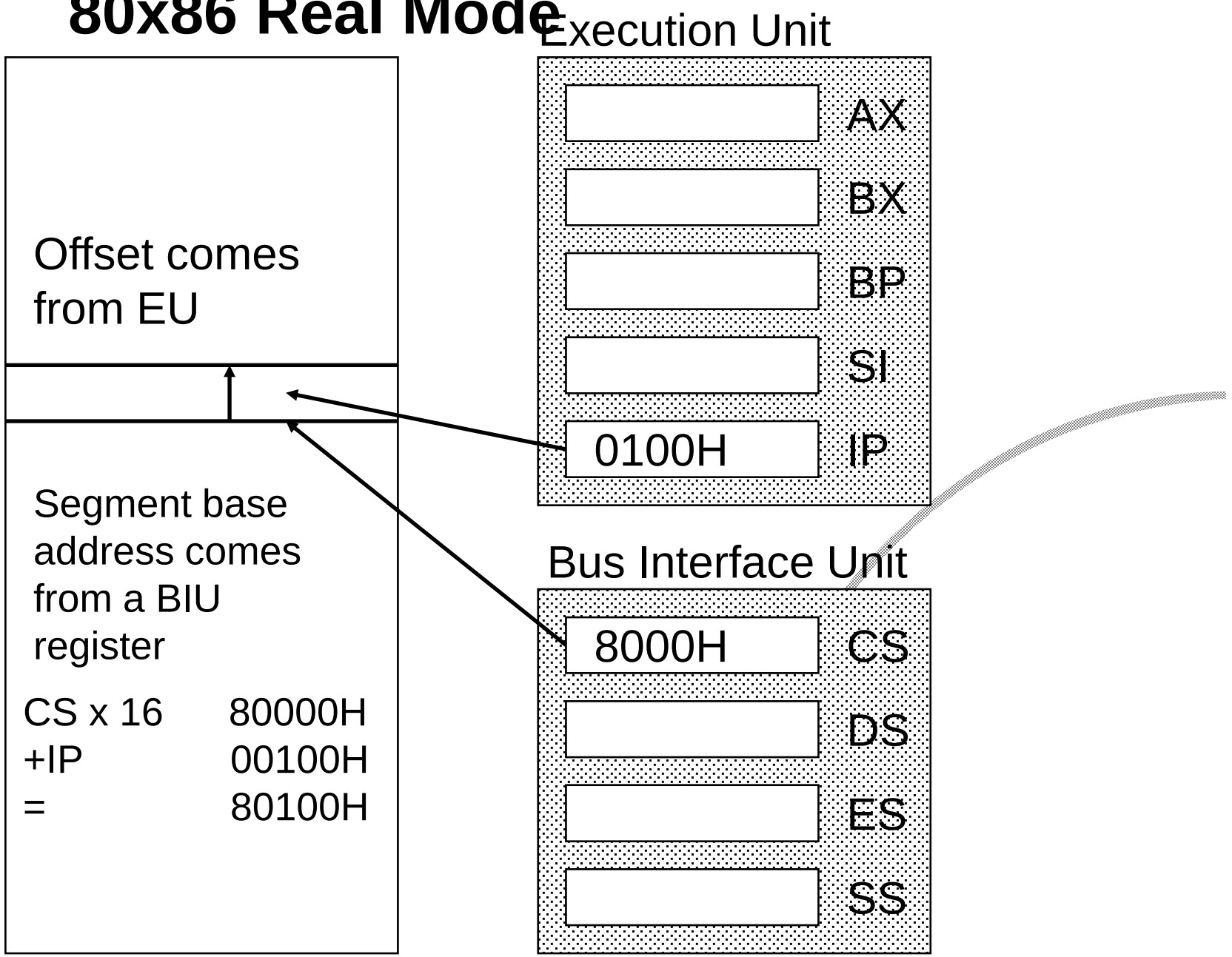
Using Visual Age C++

OS/2 is Big!

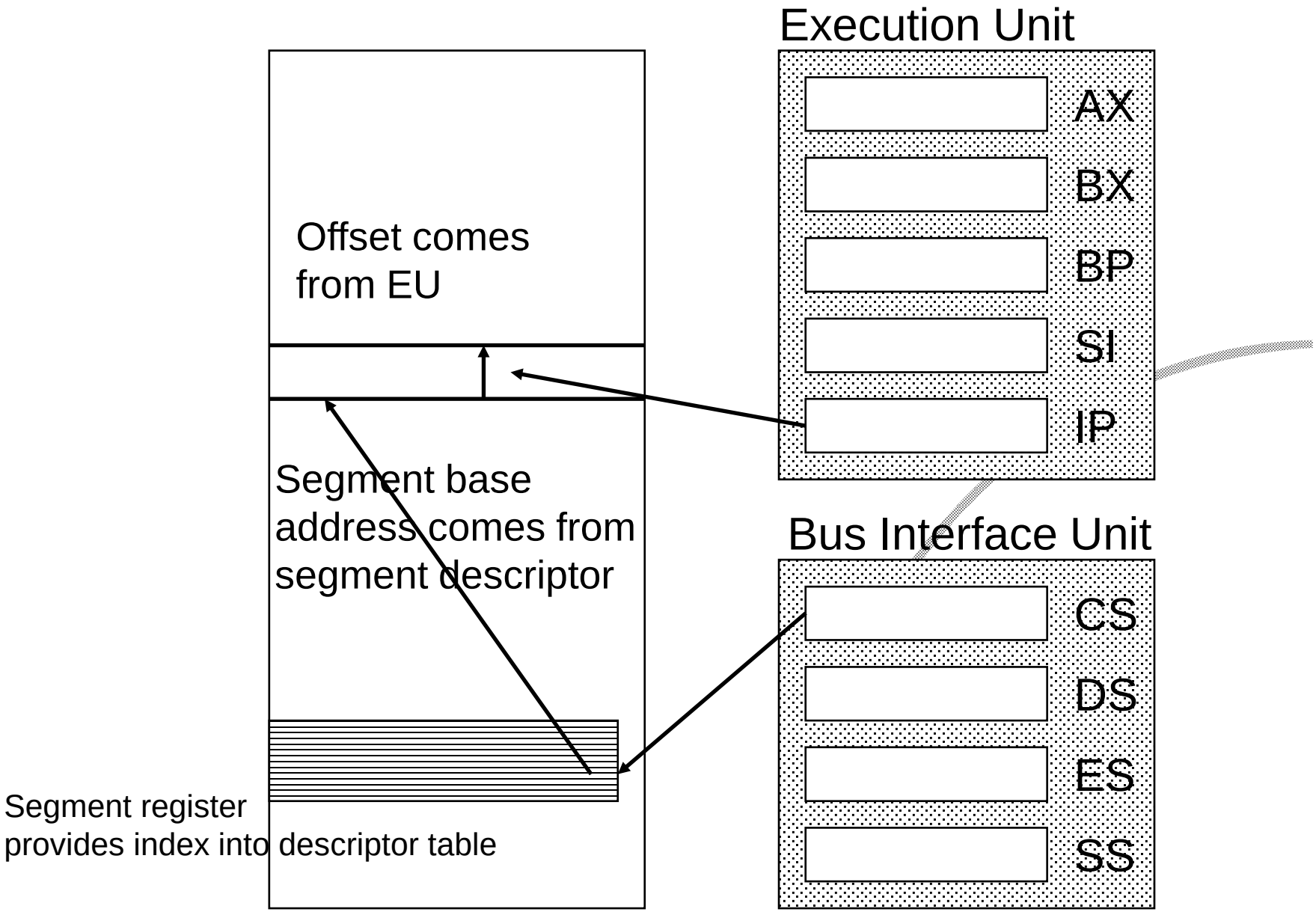
- DOS 1.0 took one man-month to write
- DOS 3.0 took a small team of programmers six months
- DOS has approximately 80 function calls

- OS/2 has had eight years of development by a large team
- OS/2 has over 1200 function calls, plus several hundred messages
- OS/2 offers many different ways to do things, because it runs on the 80286 processor in protected mode, and so
- If the operating system can't do it, then it can't be done

80x86 Real Mode



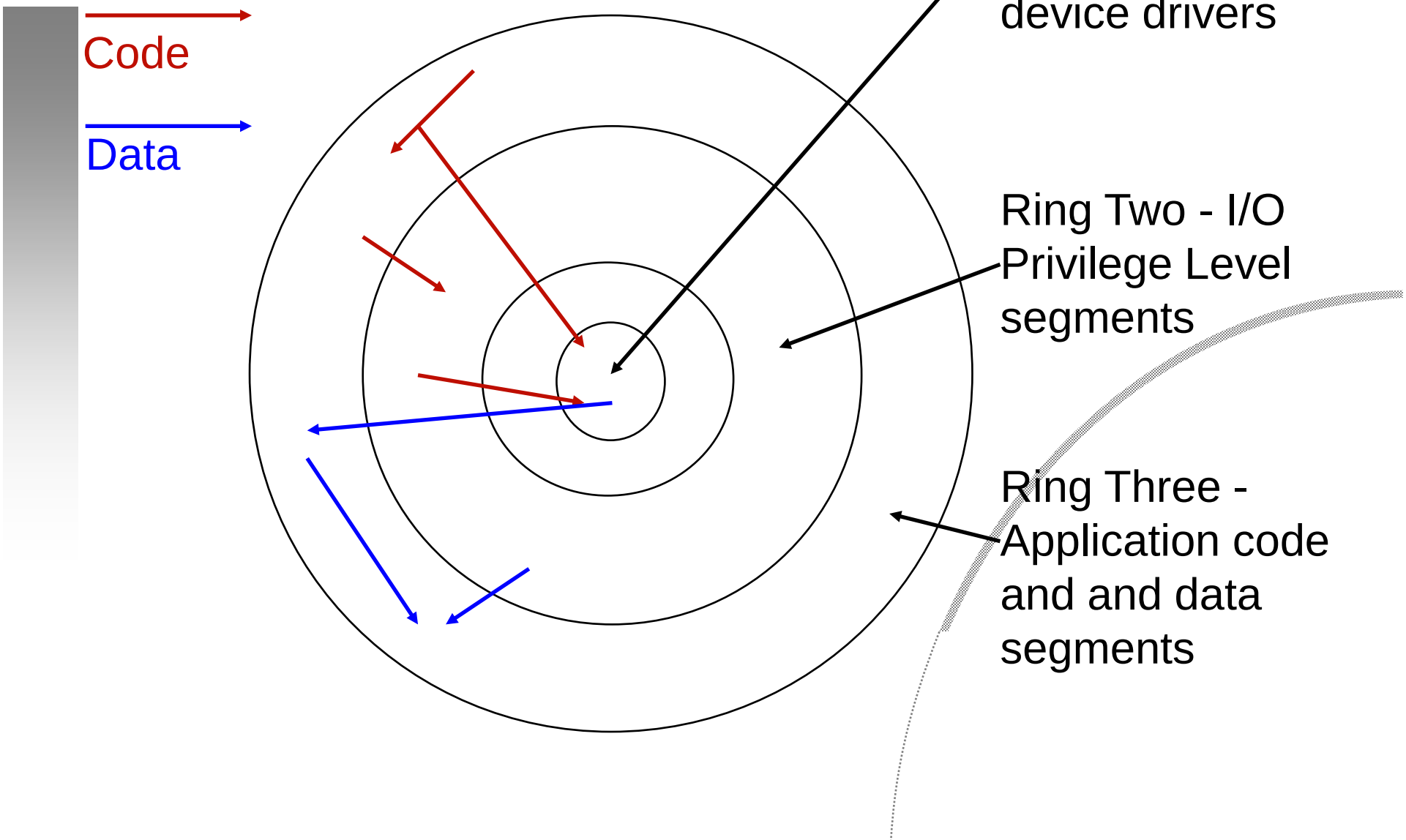
80286 Protected Mode



80286 Protected Mode Benefits

- Selector-based Addressing
  - –GDT, LDT's, IDT
- Descriptor Table Contents
  - –Segment Base
  - –Segment Limit
  - –Privilege Level
    - ►*Enforces ring-based architecture*
    - ►*Call segments at same or inner level*
    - ►*Access data at same or outer level*
    - ►*Can perform I/O iff IOPL (in processor registers) >= descriptor privilege level*
  - –Protection info: R/W, R/O, X/R, X/O

Ring-Based Architecture



Protected Mode Benefits

- Separate stacks for each ring
- Task state segments
- Virtual memory support
  - Page / segment faults
  - Interruptible instructions
  - Recoverable stack faults

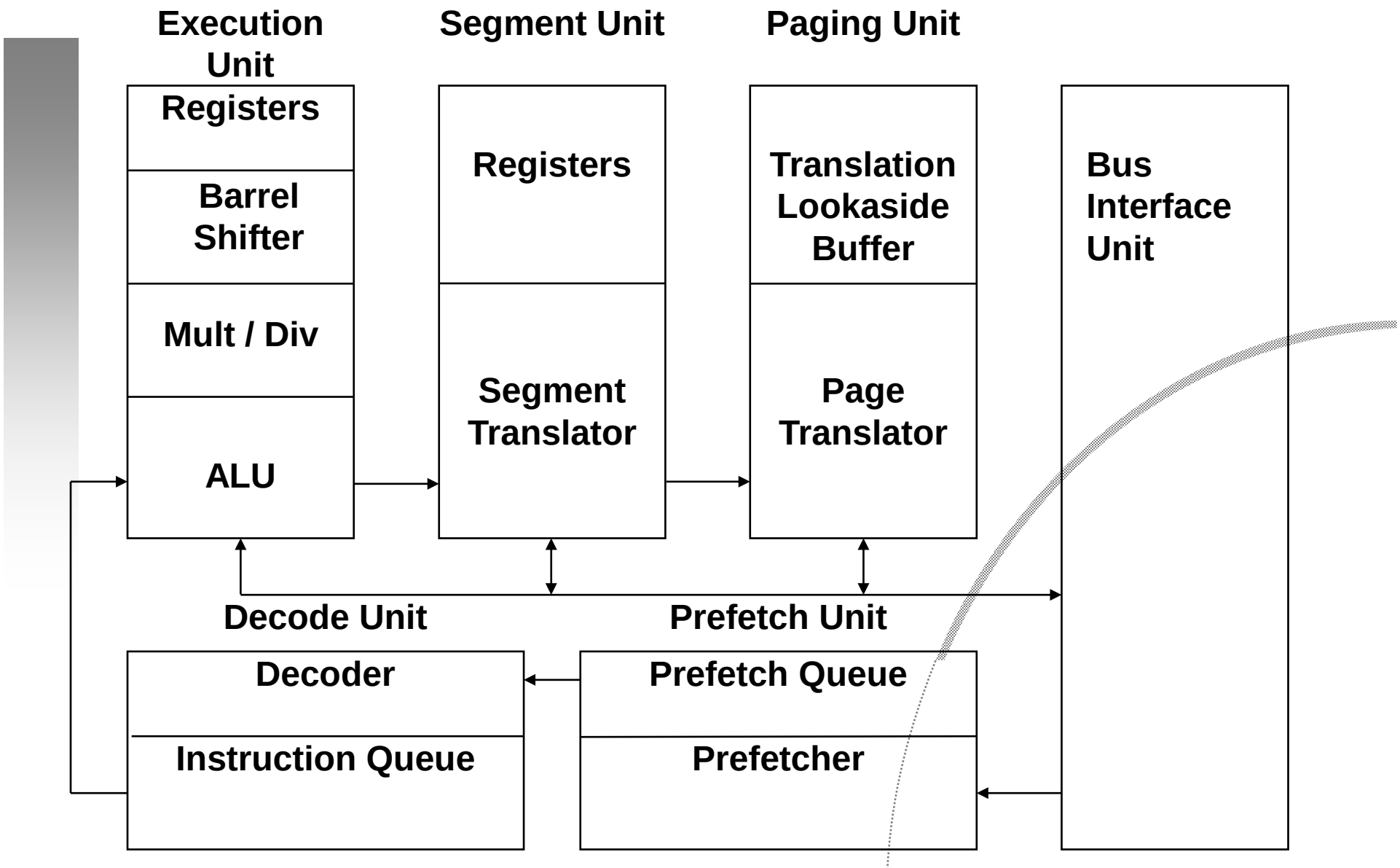
Intel 80x86 Procesor Family

- 8086 / 88
  - Real Mode Only:
  - 1 MByte physical memory (IBM PC architecture: 640 KB)
- 80286
  - Real Mode
  - Virtual Address Protected Mode:
    - *16 MBytes physical memory*
    - *1 GByte virtual memory per process*
    - *Interprocess protection through ring architecture*
- 80386 / 486
  - Real Mode
  - 286 Protected Mode
  - 386 Native Mode
  - 8086 Virtual Machine Mode

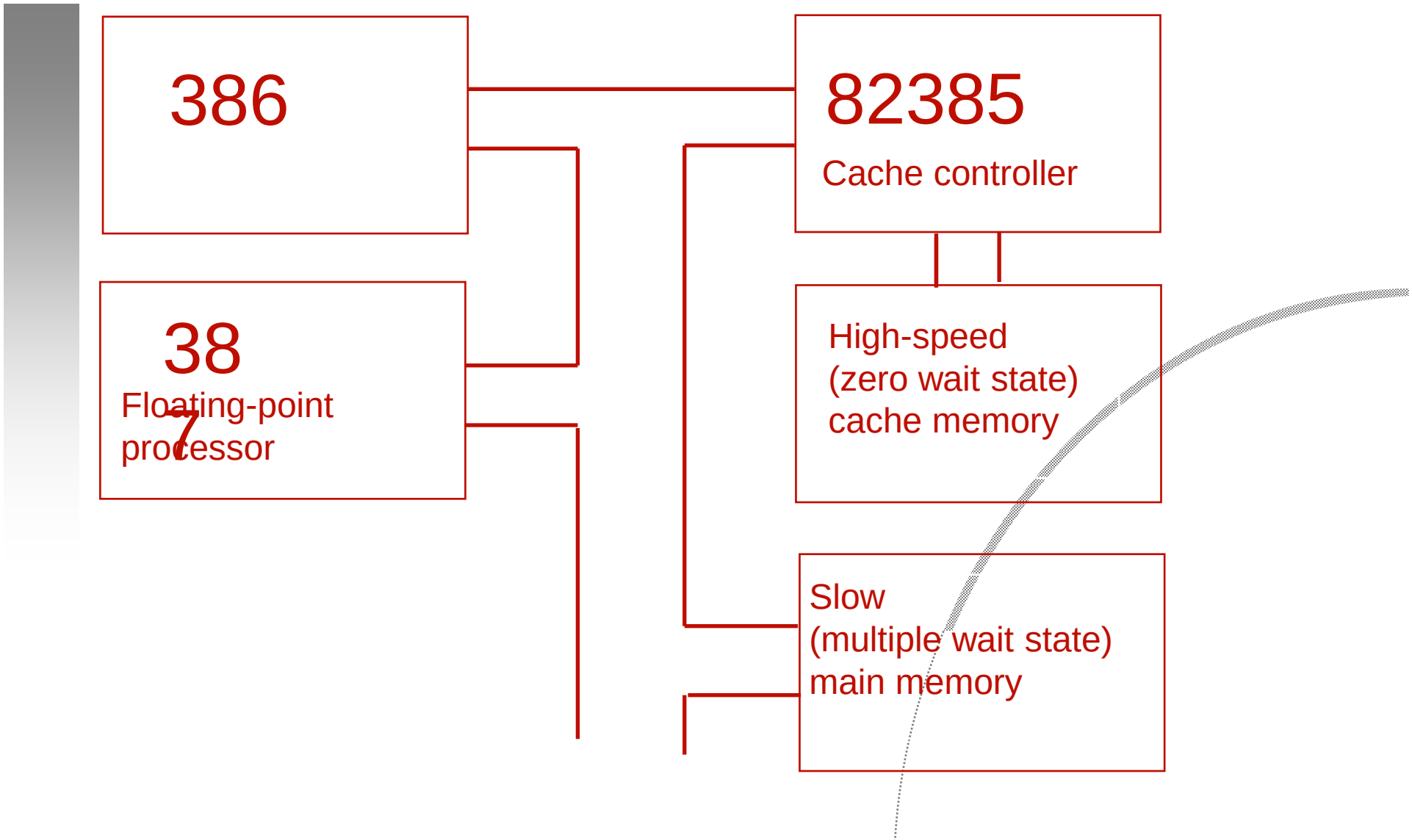
The 80386DX Processor

- 16-bit registers (AX, BX, etc.) become 32-bit (EAX, EBX)
- Crunches twice as much data per clock cycle
- Fetches twice as much data per bus cycle
- Offset comes from a 32-bit register - can be up to 4 GB
- Swaps 4KB pages for better swapper performance
- Modes:
  - –Real Mode
  - –286 Protected Mode
  - –386 Native Mode
    - ►*4 GBytes physical memory*
    - ►*64 Terabytes virtual memory per process*
  - –8086 Virtual Machine mode
- 386SX has same features, but lower performance and addressing capabilities
- 386SX is poor compromise for the money

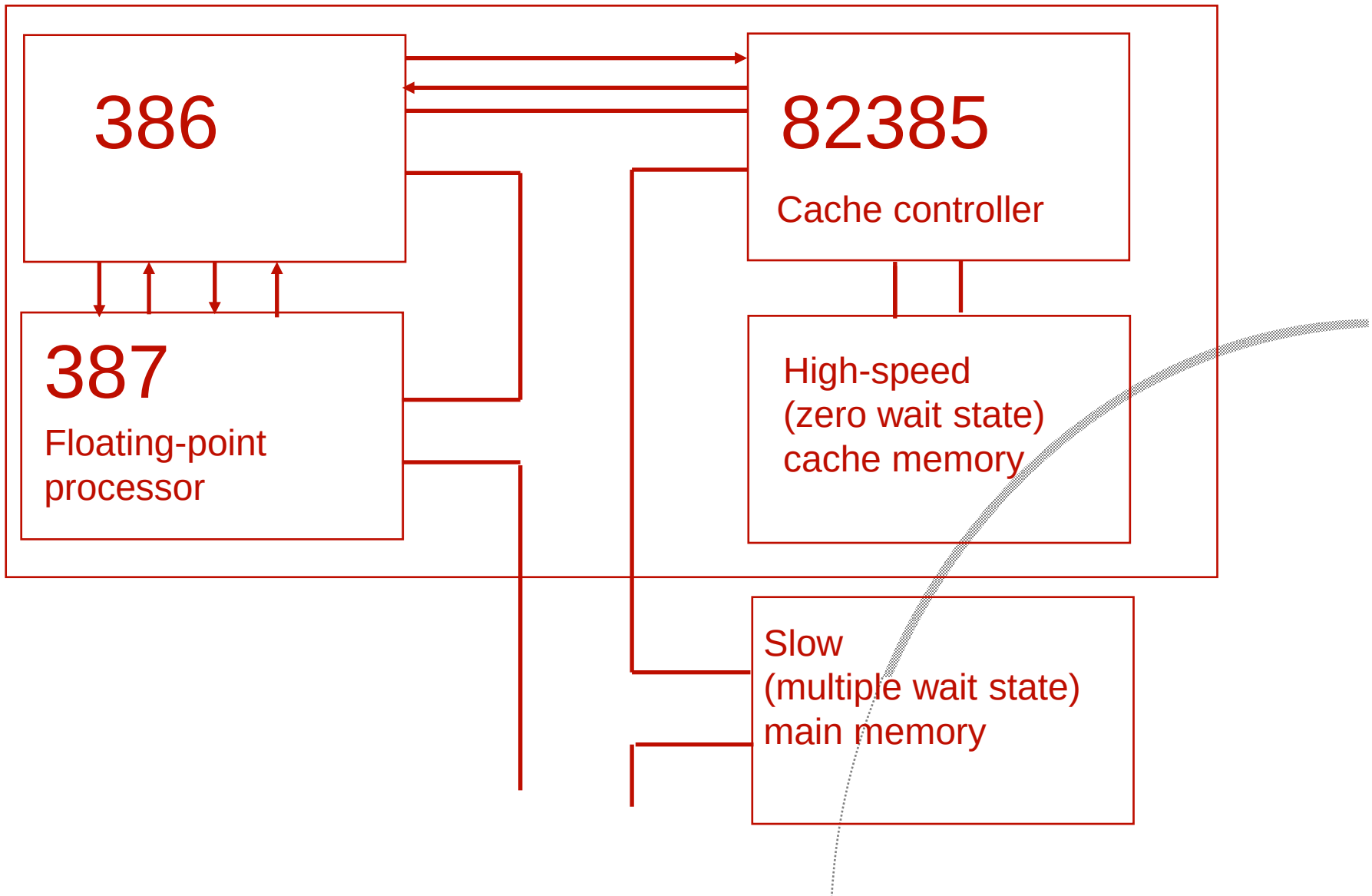
386 Block Diagram



A 386 Needs Support Chips



A 486 Doesn't


386
82385
Cache controller
387
Floating-point processor
High-speed (zero wait state) cache memory
Slow (multiple wait state) main memory

Intel Processor Performance

- 286 at 8 MHz: 1.0 MIPS
- 386 at 33 MHz: 8.0 MIPS
- 486 at 33 MHz: ~20 MIPS
- Pentium 60 - 80 MIPS
- P7 (1997):
  - 4 integer execution units
  - 2 floating point units
  - 1 digital video interactive processor
  - 2 MBytes cache on-chip
  - 1 inch square
  - 250 MHz clock speed 2,000 MIPS

Applications Cannot Perform Privileged Operations

- Address memory not set up for them in the LDT/GDT
- Edit/examine LDT/GDT
- Hook interrupt vectors
- Run at ring 0
- Call the BIOS
- Do I/O (without OS permission)
- Reference beyond a segment's length
- Put garbage in segment registers
- Execute from a data segment
- Write into a code segment (except through CSAlias)
- Perform segment arithmetic

Memory Models

- Tiny Model (DOS only)
  - –all code and data in one <64K segment
- Small Model
  - –One code segment, one data segment
- Medium Model
  - –Multiple code segments, one data segment
- Compact model
  - –One code segment, multiple data segments
- Large model
  - –Multiple code segment, multiple data segments
- Huge model
  - –As for large, but arrays can extend through multiple segments

The _near and _far keywords

- When referring, in a small or compact model program, to a function in another segment (e.g. in a DLL or in the OS kernel) it must be declared with the _far keyword.
- When referring, in a small or medium model program, to a data item in another segment (e.g. in a DLL or in a segment allocated with DosAllocSeg) it must be declared with the _far keyword.
- Remember that library functions may expect near arguments.
- The _near keyword performs a similar, but complementary function.
- If in doubt,compile in large model, and accept the resulting performance and space penalties.
- Not required for 0:32 model in OS/2 2.1

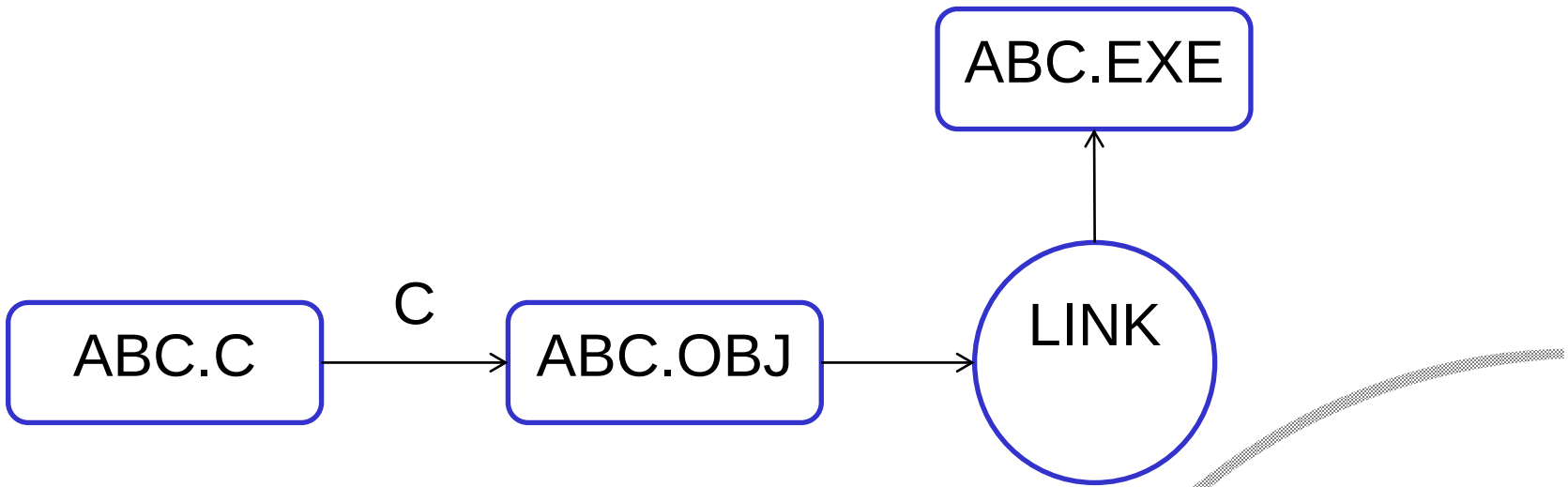
Introduction to Tools

- IBM Visual Age C++
  - 32-bit C/C++ compiler
  - Workframe 3.0 Integrated Development Environment
  - On-line docs and samples
  - Project Smarts
  - Browser, Debugger, Performance Analyzer
  - Visual Builder
  - Data Access Builder
- IBM Toolkit/2
  - RC
  - IMPLIB
  - Icon Editor
  - Font Editor
  - Dialog Editor
  - MARKEXE

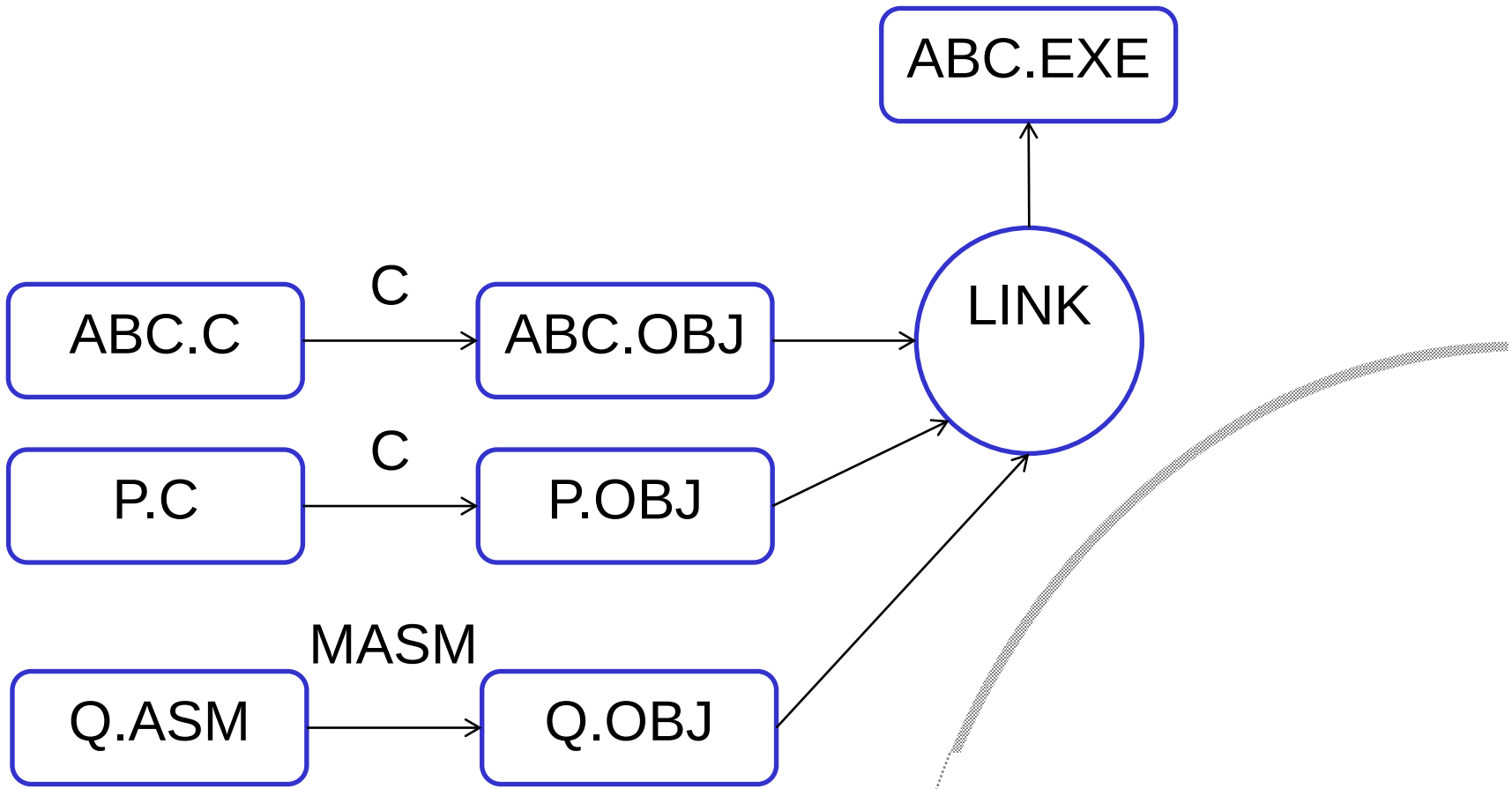
IBM Visual Age C++ Compiler

- Industrial-strength 32-bit 386/486/Pentium compiler
- Cannot generate 8086/286 DOS/Windows/OS/2 1.3 code (use Watcom C/C++ for this)
- Based on AIX C compiler with 386 code generator
- Multiple parameter-passing techniques
  - _System for OS/2 API's
  - _Optlink passes parameters in processor registers where possible for speed
- Several function libraries
  - Conventional
  - Multithread
  - Subsystem
- Direct-To-SOM Code Generation from C++ classes
- Package includes Toolkit and Workframe

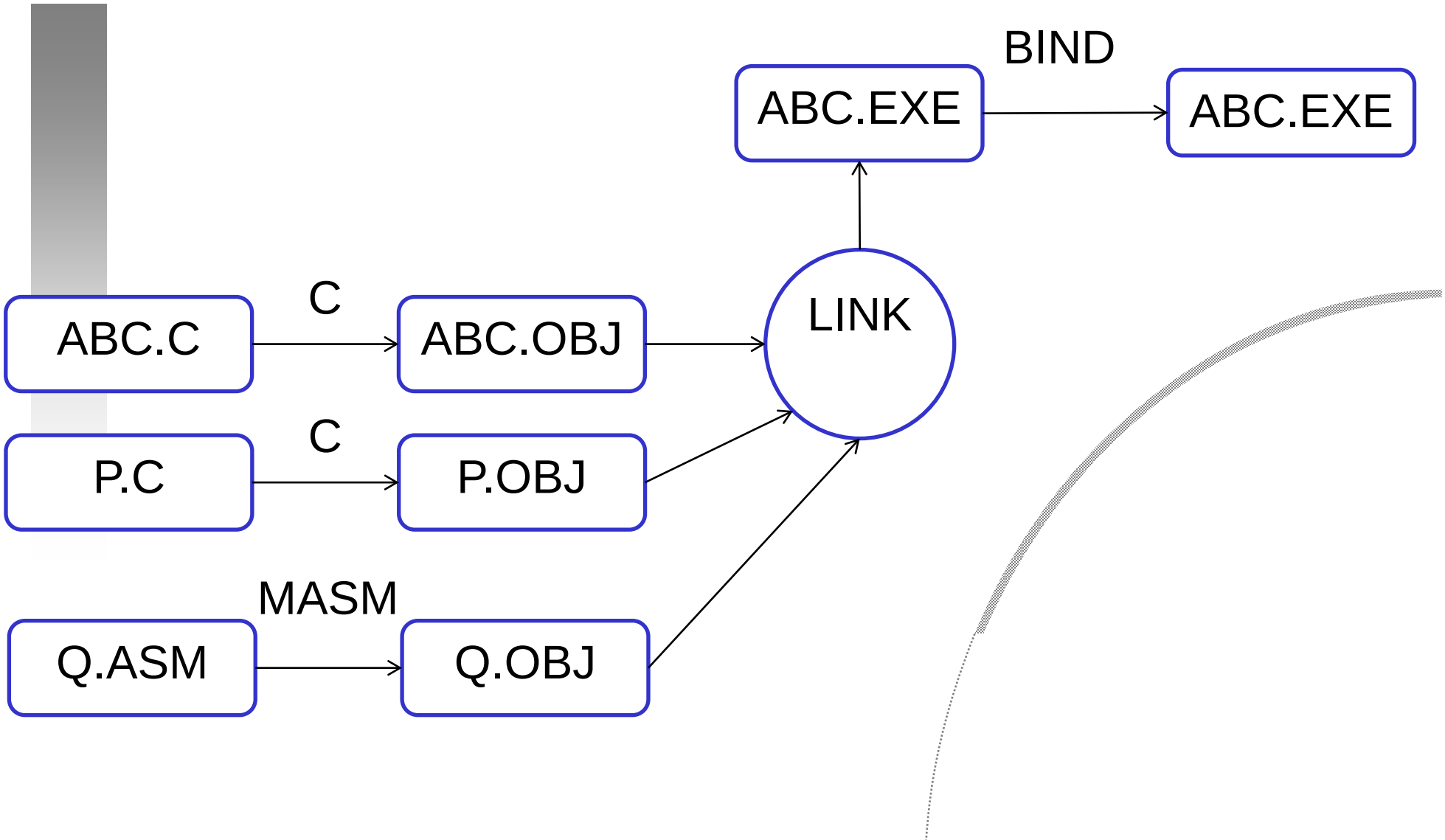
Basic OS/2 Compile



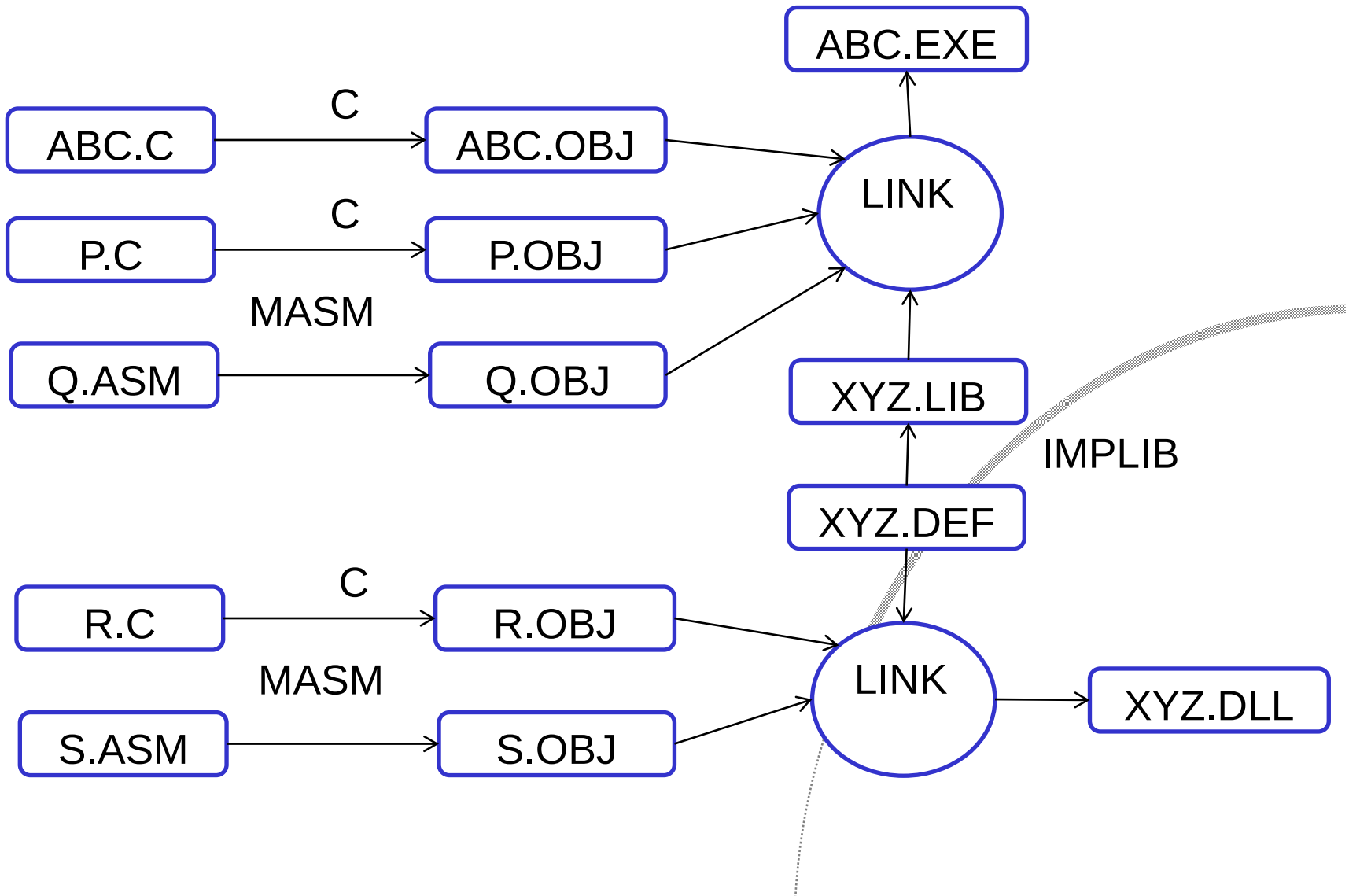
Larger OS/2 Compile



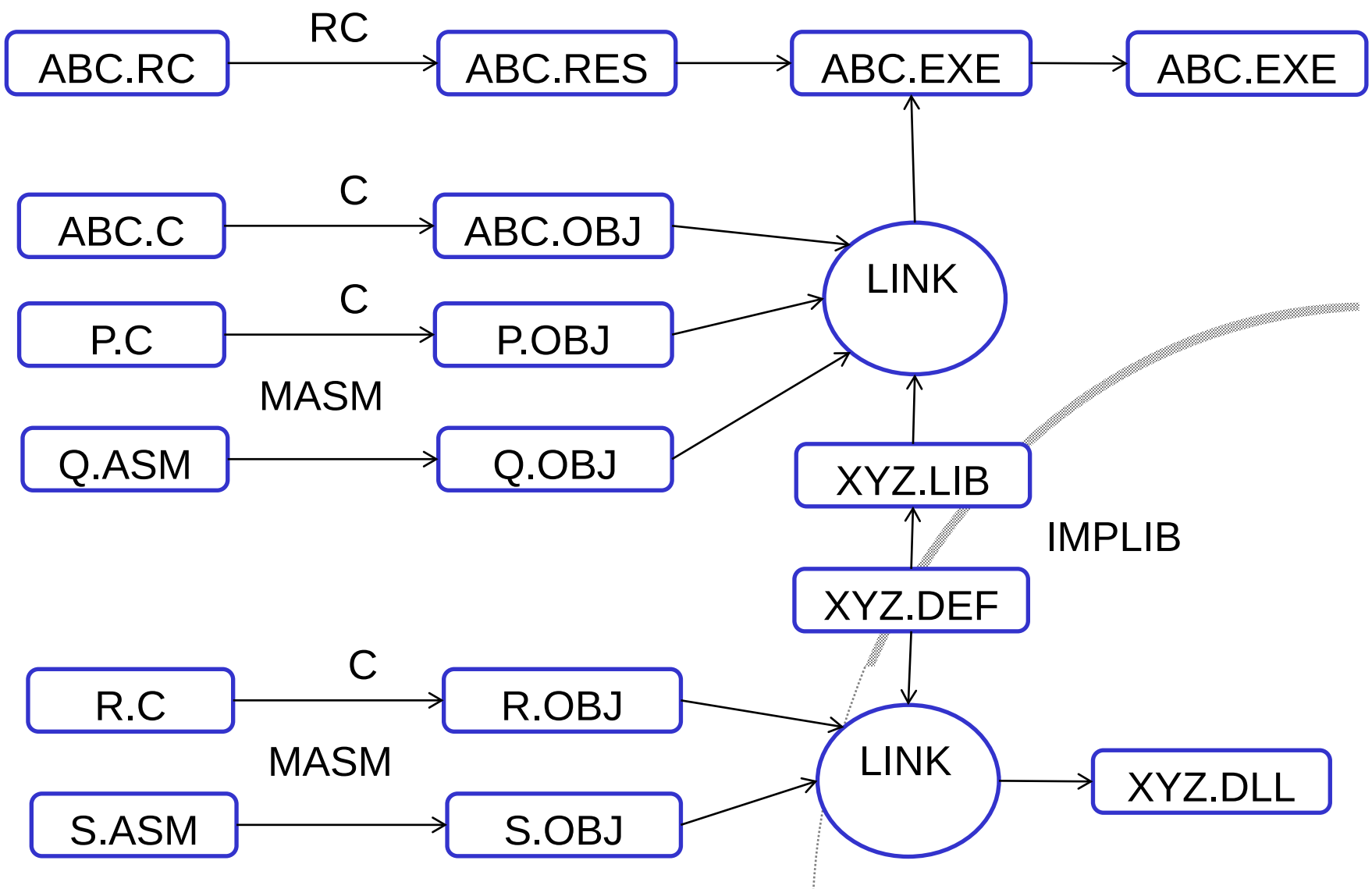
Bound FAPI Application



Creating and Using a DLL



A Full PM Application



Project Make Files

- Projects typically comprise multiple files (.C, .RC, .ASM, .DEF, etc)
- These need to be recompiled and linked in the correct sequence
- But not all every time
- The steps required to recreate the target executable are recorded in a file called a Make file (.MAK)
- This is used by the NMAKE utility to drive the appropriate utilities
- The WorkFrame can create Make files automatically
- Changes to compile and link options will require regeneration of the make file

The EPM Editor

- Needs some reconfiguration
  - Options / Preferences / Settings...
    - *Tabs to 4, Font to Courier Bitmap 13 x 8*
  - Options / Preferences / Ring Enabled
  - Options / Save Options
- But is otherwise quite usable (with practice)
- Edit multiple files in one ring, using File / Add File (F8)
- Save work with File / Save (F2) or Close (F4) or Quit (F3 - no save)
- Command Box (Ctrl - I)
- Common commands
  - EXPAND OFF
- Macro language: REXX (of course)
- Compiler submenu activated when invoked from compiler error output window

Day 1 – Session 2

Introduction to PM

Introduction to Presentation Manager

OS/2's Windowing System

Presentation Manager Features

- Integrated Component of OS/2
- Standard full-colour graphical interface
- New user interface replaces or augments CMD.EXE and COMMAND.COM
- Non-PM, VIO, applications can run in a window
- High level graphics language / library (GPI)
- API and OEM layers quite different from Windows

Key Concepts

- Screen Group
  - –is the basic unit of I/O management
  - –is a virtualization of the console device
    - ►*keyboard*
    - ►*screen*
    - ►*mouse*
- Windowing
  - –is the means by which console input and output of the PM screen group is multiplexed
  - –Each window is a virtual sub-console of the PM screen group
- Message processing
  - –is the mechanism chosen to enforce input serialization and output synchronization among windows

Handles

- As in the DOS file system, OS/2 and PM objects have handles
- A handle is a 32-bit value which may be the adress or array index of the corresponding data structure
- Handles are only known at run-time
- Do not assume what a handle represents
- Objects which have handles:
  - Windows
  - Heaps
  - Files
  - Pipes
  - Queues
  - Semaphores
  - You could think of addresses as memory object handles

Presentation Manager Services

- Window Management
- Window Classes (Private and Public - predefined windows)
- Message Queues
- Menus
- Dialog Windows
- Icons
- Cursors
- Graphics Output
- User Messages
- Mouse Input
- Keyboard Input

What is a Window?

- A window is a complex data structure in memory.
- Presentation Manager will (may) give an on-screen appearance which corresponds to this data structure.
- Associated with the window will be a window procedure which provides the window behaviour and appearance
- This window procedure is also known as the window class
- Several types of window classes are available:
  - Frame windows
  - Control windows
  - Client and dialog windows (coded by the application programmer)

Windows

- Each Presentation Manager window has a handle
- Window handles are of type HWND
- Obtained as the return value of WinCreateWindow()
- or returned value and referenced parameter to WinCreateStdWindow()
- The desktop window handle is the manifest constant HWND_DESKTOP
- The desktop object window handle is the manifest constant HWND_OBJECT

Getting Window Handles

- To find the handle of a window you didn't create, use
- WinQueryWindow(hwnd,) for parent, owner, next, previous
- WinWindowFromID(hwnd, ID) for a component such as a pushbutton, list box, etc
- WinBeginEnumWindows() to traverse a window's parentage tree
- WinWindowFromPoint() when you know a screen or window coordinate and need to find the handle of the window under that point.

Frame Windows

- Are 'containers' which put other application window components in the correct relative positions and control them
- Have special properties
  - Destroy all owned windows, even if they are not descendants, when the frame is destroyed
  - Moves owned windows when the frame is moved
  - Changes the Z-order of all owned windows when the frame window changes
  - Hides all owned windows when the frame window is minimized or hidden
- If an application window needs this behaviour, it must be explicitly coded into the window procedure.

Presentation Manager Windows

- What the user sees as a 'window' is actually many windows
- Each window component is a predefined winproc
- Standard winprocs:
  - Frame
  - Border
  - System Menu
  - Title Bar
  - Minimise / maximise box
  - Application Menu
  - Vertical and Horizontal scroll bars
  - Control windows (buttons, entry fields, etc)
- Client window - the white area in the middle
- The behaviour of the winproc, i.e. its response to input, defines the window appearance.

Specifying Frame Components

- HWND hwndFrame;
- FRAMECDATA fcData;

- fcData.cb = sizeof(FRAMECDATA);
- fcData.flCreateFlags = FCF_MENU | FCF_TITLEBAR |
- FCF_MINMAX | FCF_ICON |
- FCF_SYSMENU |
- FCF_SIZEBORDER;

- fcData.hmodResources = (HMODULE) 0;
- fcData.idResources = ID_MAIN;

Creating a Frame Window

- hwndFrame = WinCreateWindow(
  - HWND_DESKTOP, /* Parent window */
  - WC_FRAME, /* Window Class */
  - (PSZ)NULL, /* Window Text */
  - 0L, /* Window Styles */
  - 0, /* Bottom Left x */
  - 0, /* Bottom Left y */
  - 0, /* Width */
  - 0, /* Height */
  - (HWND) 0, * Owner window handle */
  - HWND_TOP, /* Z-order */
  - ID_MAIN, /* Window ID */
  - &fcData, /* Creation Data */
  - NULL); /* PRES PARAMS */

Creating a Client Window

- HWND hwndClient;
- hwndClient = WinCreateWindow(
  - hwndFrame, /* Parent window */
  - (PSZ)WC_MYCLASS, /* class name */
  - PSZ(NULL), /* Window text */
  - 0L, /* Window Styles */
  - 0, /* Bottom Left x */
  - 0, /* Bottom Left y */
  - 0, /* Width */
  - 0, /* Height */
  - (HWND)0, /* Handle to owner */
  - HWND_TOP, /* Z-order */
  - FID_CLIENT, /* ID */
  - NULL, /* Control Data */
  - NULL); /* Pres Params */

WinCreateStdWindow

- Most applications need both a frame and client windows, and create both by calling WinCreateStdWindow().
- Note how the hwndFrame and hwndClient are both returned:
- ULONG ctldata = FCF_STANDARD | FCF_VERTSCROLL;
- hwndFrame = WinCreateStdWindow(HWND_DESKTOP,
- WS_VISIBLE,
- &ctldata,
- (PSZ)WC_CLIENTCLASS,
- (PSZ)&TitleBarText,
- WS_VISIBLE,
- (HMODULE)0,
- ID_RESOURCE,
- &hwndClient);

A PM Process

- Consists of:
  - its screen group
  - handles (file handles, window handles, queue handles)
  - local descriptor table
  - control thread
- An application which wishes to create a window registers a procedure (with the WinRegisterClass() function) to perform output and accept input for the window. This is the window 'class'.

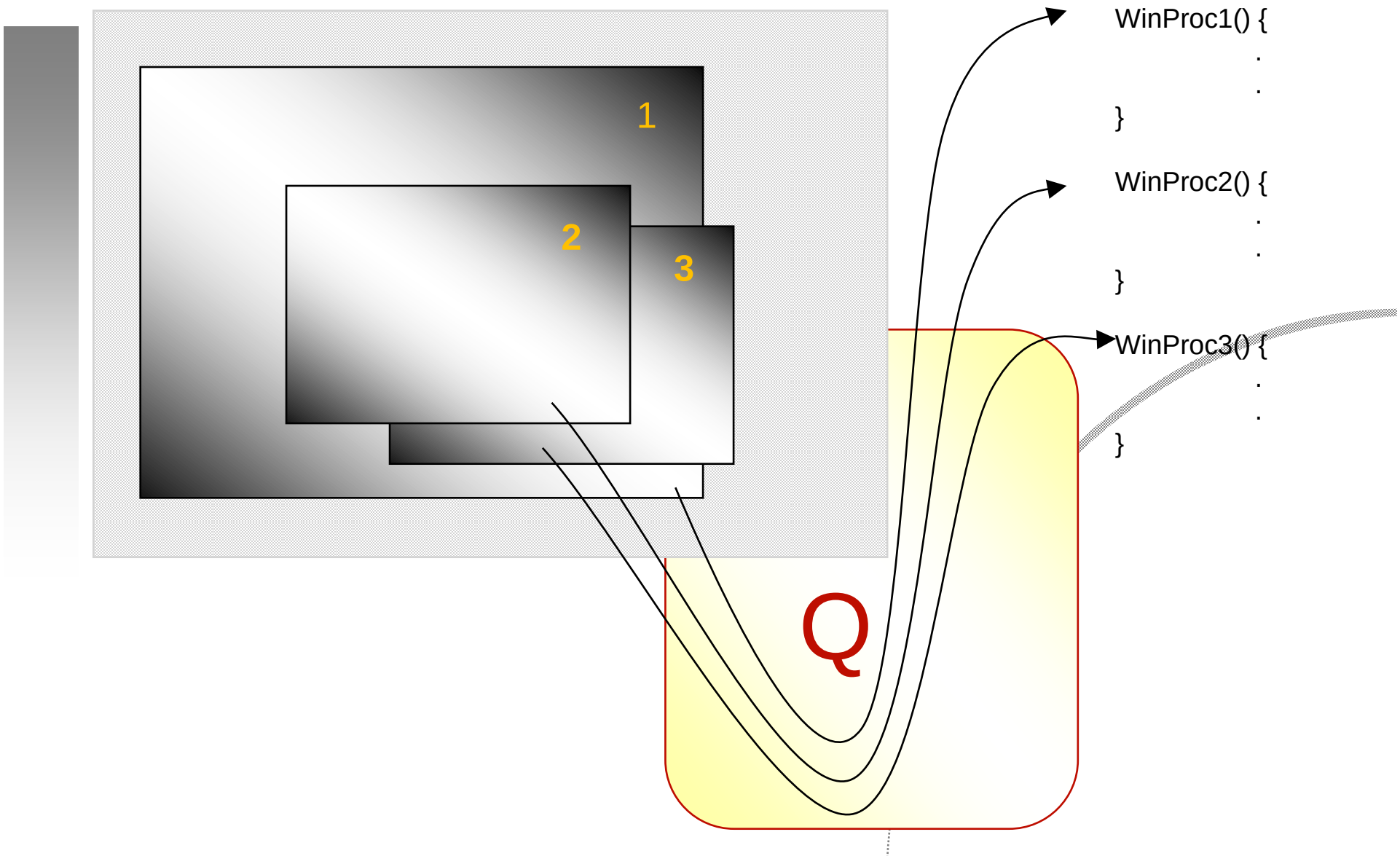
Logic of a PM Process (cont)

- Before the application creates a window, it creates a message queue (with the WinCreateMsgQueue() function) to serialize input. This is because
- Mouse motions and keystrokes must be processed in the correct order.
- For example, File Save must be processed before Exit
- All input to a PM process comes through its input queue
- This is what defines a PM process or thread.

Logic of a PM Process (cont)

- The process creates a window using its registered 'class', i.e. its WinProc
- Input is directed to a window, and is placed on the message queue of the process that created the window. The message then is subsequently dispatched to the WinProc responsible for the window real estate

Logic of a PM process (cont)



Logic of a PM Process (cont)

- PM applications are event-driven
- Events are dispatched using messages
- Events which generate messages:
  - Keyboard state changes
  - Mouse state changes
  - Timer state changes
  - Window state changes
  - Clipboard state changes
- Messages are sent to threads, not windows
- Once delivered to a thread, the message is dispatched to the window
- For each window, there is an input procedure

What's a Message?

```
/* QMSG structure */
typedef struct _QMSG    /* qmsg */
{
   HWND    hwnd;        /* Target window handle */
   ULONG   msg;         /* WM_PAINT,  WM_COMMAND, etc */
   MPARAM  mp1;         /* menu sel, sb click, etc. */
   MPARAM  mp2;         /* sb info, dlgbox text, etc */
   ULONG   time;
   POINTL  ptl;         /* Mouse position */
   ULONG   reserved;
} QMSG;
typedef QMSG *PQMSG;
```

Logic of a PM Process (cont)

- A Presentation manager application is a process that:
  - has a thread that polls its message queue for events
  - may have one or more threads to processes events
- If the application creates a window
  - it registers a procedure to service events for the window
  - this is the 'window procedure' or winproc.

More on Messages

- PM applications can elect to send messages
  - –Asynchronously, via WinPostMsg (message placed on event queue, immediate return)
  - –Synchronously, via WinSendMsg (receiver processes message before return)
- PM applications get messages
  - –Asynchronously, via WinGetMsg
  - –Synchronously, via direct call to their winprocs

Day 1 – Session 3

Lab Exercise 1

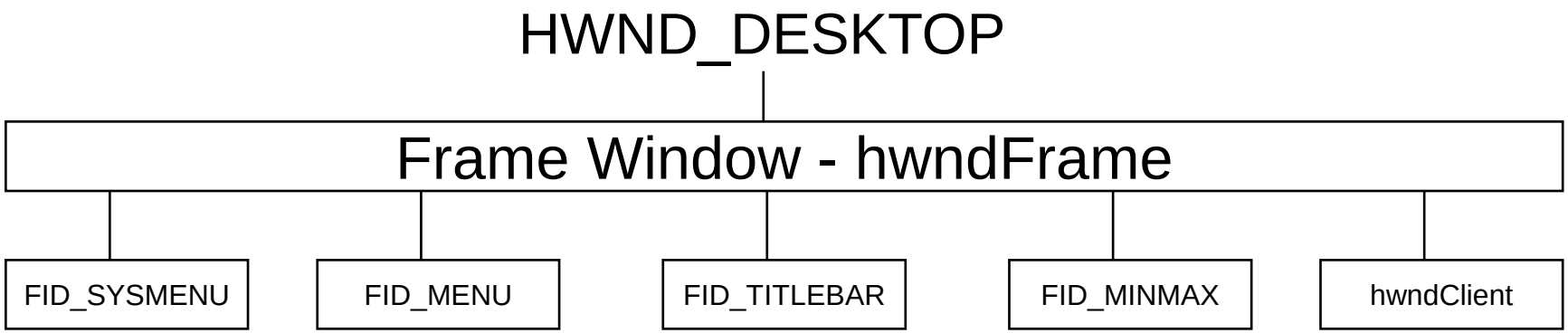
Day 1 – Session 4

Window Parentage and Ownership

Window Parentage and Ownership

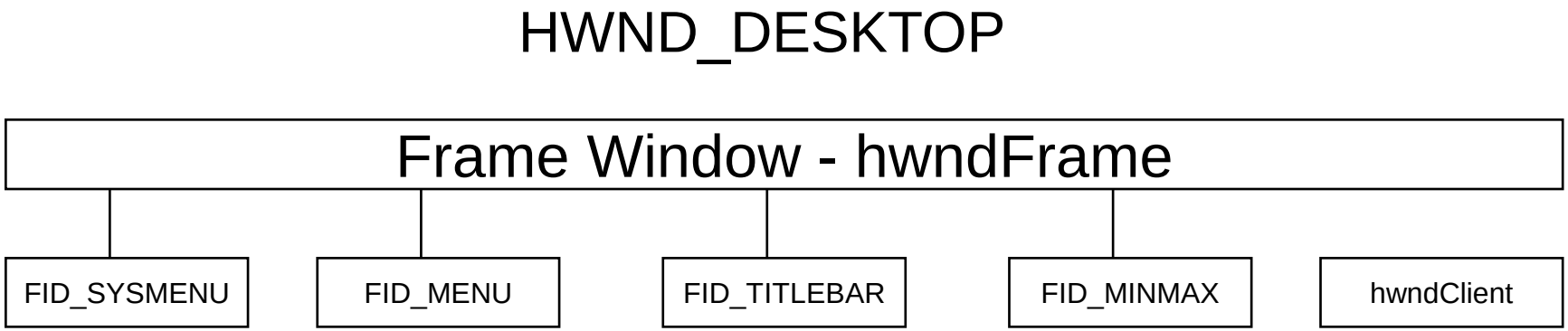
Window Message Passing

Window Relationships - Parentage



- Windows are painted on the screen relative to their parents (x, y position, size and z-order)
- You don't know the handles of the other children, just their predefined ID's, which allows you to retrieve the handles with
- WinWindowFromID(hwndFrame, FID__??)

Window Relationships - Ownership



- Windows notify their owners of significant events by sending WM_CONTROL messages.
- A window need not have any owner.
- Ownership is set by one of the parameters to the WinCreateWindow() call
- or by WinSetOwner()

The Frame's Children

• Sys Menu	WC_MENU	FID_SYSMENU
• Title Bar	WC_TITLEBAR	FID_TITLEBAR
• MinMax	WC_MENU	FID_MINMAX
• Menu Bar	WC_MENU	FID_MENU
• Horz Scroll Bar	WC_SCROLLBAR	FID_HORZSCROLL
• Vert Scroll Bar	WC_SCROLLBAR	FID_VERTSCROLL
• Client Window Area	?	FID_CLIENT

- The client window is not necessarily visible, but may simply coordinate some children which completely fill the client window area.

Control Windows

- PM provides many types of predefined control windows:
  - –WC_SCROLLBAR
  - –WC_STATIC
  - –WC_ENTRYFIELD
  - –WC_MLE
  - –WC_LISTBOX
  - –WC_COMBOBOX
  - –WC_BUTTON
  - –WC_SPINBUTTON
  - –WC_VALUESET (OS/2 2.0)
  - –WC_NOTEBOOK (OS/2 2.0)
  - –WC_CONTAINER (OS/2 2.0)
  - –WC_SLIDER (OS/2 2.0)

What's a Message?

```
/* QMSG structure */
typedef struct _QMSG    /* qmsg */
{
   HWND    hwnd;        /* Target window handle */
   ULONG   msg;         /* WM_COMMAND, WM_CONTROL, etc */
   MPARAM  mp1;
   MPARAM  mp2;
   ULONG   time;
   POINTL  ptl;
   ULONG   reserved;
} QMSG;
typedef QMSG *PQMSG;
```

Packers and Crackers

- **Packers**
- MPFROM2SHORT(s1, s2)
- MPFROMCHAR(ch)
- MPFROMHWND(hwnd)
- MPFROMLONG(l)
- MPFROMP(p)
- MPFROMSH2CH(s, uch1, uch2)
- MPFROMSHORT(s)
- MRFROM2SHORT(s1, s2)
- MRFROMLONG(l)
- MRFROMP(p)
- MRFROMSHORT(s)

- **Crackers**
- SHORT1FROMMP(mp)
- SHORT2FROMMP(mp)
- CHAR1FROMMP(mp)
- CHAR2FROMMP(mp)
- CHAR3FROMMP(mp)
- CHAR4FROMMP(mp)
- HWNDFROMMP(mp)
- LONGFROMMP(mp)
- PVOIDFROMMP(mp)
- PVOIDFROMMR(mr)
- LONGFROMMR(mr)
- SHORT1FROMMR(mr)
- SHORT2FROMMR(mr)

Notes on Messages

- Use WinSendMsg() when you want guaranteed receipt and a returned value
- If the recipient of WinSendMsg() hangs, you hang too.
- WinSendMsg() is effectively an indirect function call
- Use WinPostMsg() when you can wait on processing of the message
- Use WinPostMsg() to post a WM_QUIT message to yourself to end your thread or process
- Posting or sending one message may cause a chain reaction of many message movements
- Be careful not to Post someone a pointer to data and then free the memory before thay can refer to it.

Messages

- Messages are classed by a prefix:
  - –WM_ Window message
  - –SBM_ Scroll-bar message
  - –BM_ Button message
  - –EM_ Entry-field message
- Some messages are used to 'program' child windows - set text, position, size, etc
- Some messages (WM_COMMAND, WM_HSCROLL, WM_CONTROL with notification code) are sent by control windows to notify owner windows of events

Useful Messages

- WM_CREATE
  - Sent to a window when it is being created. Used to perform initialisation, creation of child windows
  - pCtlData = (PVOID) PVOIDFROMMP(mp1)
  - pcrst = (PCREATESTRUCT) PVOIDFROMMP(mp2)
- WM_INITDLG
  - Sent to a dialog window when it is being created. Analogous to WM_CREATE.

More Useful Messages

- WM_PAINT
  - Sent when a window is to be repainted. Does not use any parameters. An application should return zero if it processes this message.
- WM_COMMAND
  - Sent to a window when it has a command to process or when a keystroke has been translated into a comman by an accelerator table
  - usCmd = (USHORT) SHORT1FROMMP(mp1)
  - fsSource = (USHORT) SHORT1FROMMP(mp2)
    - *CMDSRC_ACCELERATOR, CMDSRC_MENU, CMDSRC_PUSHBUTTON, CMDSRC_OTHER*
  - fPointer = (BOOL) SHORT2FROMMP(mp2)
    - *True for mouse operation, false for keyboard*

More Useful Messages

- WM_CLOSE
  - Sent as a signal that the window or its application should terminate. If passed to the WinDefWindowProc function, it in turn posts a WM_QUIT message
  - Consider using this to ask the user if he wants to save changes
- WM_QUIT
  - When read by the WinGetMsg function call, causes it to return FALSE and exit the event loop

More Useful Messages

- WM_CONTROL
  - Sent when a control window wishes to report an event to its owner
  - id = (USHORT) SHORT1FROMMP(mp1)
    - *Identifies the control window*
  - usNotifyCode = (USHORT) SHORT2FROMMP(mp1)
    - *LN_ENTER, LN_SELECT, EN_CHANGE, EN_SETFOCUS, etc.*
  - usData = (ULONG) LONGFROMMP(mp2)
    - *Control- and event-specific data*

Menus

- Do not consist of procedural code in the application
- Are trees of windows, of class WC_MENU
- Are not usually created by WinCreateWindow, but are
- Loaded from the application .EXE or .DLL file resources
- When processing WinCreateStdWindow()
- Are created by the Resource Compiler

A Typical Short Menu

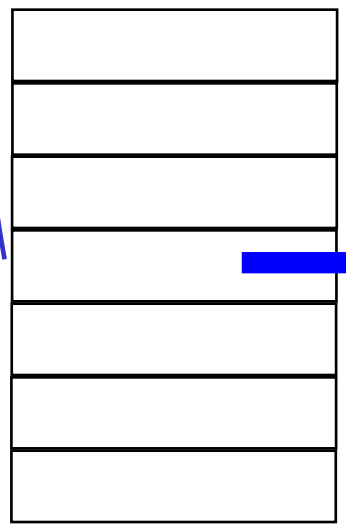
```
MENU ID_RESOURCE
BEGIN
        SUBMENU "~File", IDM_FILE
        BEGIN
                MENUITEM "~New", IDM_NEWFILE
                MENUITEM "~Open.. .", IDM_OPENFILE
                MENUITEM SEPARATOR
                MENUITEM "~Save", IDM_SAVE
        END
        SUBMENU "~Edit", IDM_EDIT
        BEGIN
                MENUITEM "~Undo\tAlt+Backspace", IDM_UNDO
                MENUITEM "Cu~t\tShift+Del", IDM_CUT
        END
END
```

The Model-View-Controller Approach

WM_COMMAND

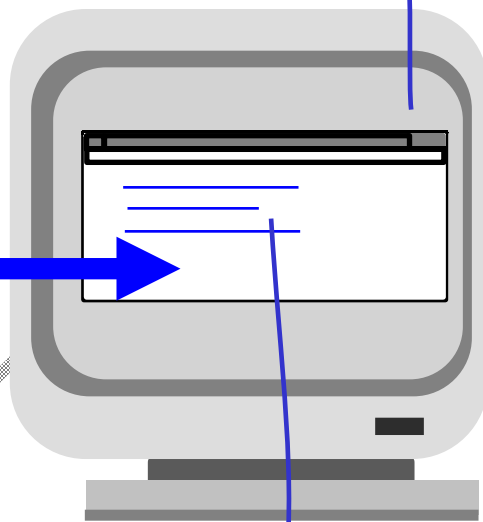
```
case ID_CLEAR:
  irbd.x = 0;
  irbd.y = 0;
  WinInvalidateRect();
  return 0;
  break;
```

WM_PAINT



Internal Representation of Business Data

```
WinBeginPaint()
WinDrawText();
GpiCharString()
.
etc
.
WinEndPaint()
```



Agenda

- Day 1
  - Session 1 – Introduction to Tools
  - Session 2 – Introduction to PM
  - Session 3 – Lab Exercise 1
  - Session 4 – Windows Parentage and Ownership
- Day 2
  - Session 1 – Window Controls
  - Session 2 – Lab Exercise 2 – Menus and Messages
  - Session 3 – Memory Management
  - Session 3 – Lab Exercise 4 – Memory Management
  - Session 4 – Dynamic Link Libraries
  - Session 4 – Lab Exercise 5 – Dynamic Link Libraries

Agenda

- Day 3
  - Session 1 – Threads, IPC and File I/O
  - Session 2 – Lab Exercise 6 - Threads
  - Session 3 - Workshop
  - Session 4 – Filesystems % EA's
  - Session 4 – Lab Exercise 8 – Directory Listing
- Day 4
  - Session 1 – Window Words, Subclassing, Dialogs
  - Session 2 – Lab Exercise 9 – Multiple Windows and Instance Data
  - Session 3 – Lab Exercise 9 continues
  - Session 4 – Standard Dialogs and INI files
- Day 5
  - Session 1 – Graphics Programming Interfase
  - Session 2 - Workshop
  - Session 3 – SOM and WPS
  - Session 4 – It's Friday…

Day 2 – Session 1

Window Controls

Window Controls

& Control Windows

Control Windows

- Are predefined window classes in the system
- WC_SCROLLBAR
- WC_STATIC
- WC_ENTRYFIELD
- WC_MLE
- WC_LISTBOX
- WC_COMBOBOX
- WC_BUTTON
- WC_VALUESET (OS/2 2.0)
- WC_NOTEBOOK (OS/2 2.0)
- WC_CONTAINER (OS/2 2.0)
- WC_SLIDER (OS/2 2.0)
- WC_SPINBUTTON (OS/2 1.3)

Scrollbars

- Obtain scroll bar handles with
  - –hwndHorzScroll = WinWindowFromID(hwndFrame, FID_HORZSCROLL)
  - –hwndVertScroll = WinWindowFromID(hwndFrame, FID_VERTSCROLL)
- Create with WinCreateWindow()
- Set the range and position with WinSendMsg(hwndScroll, SBM_SETSCROLLBAR, . .)
- Read the scrollbar with this construction:
  - –usSliderPos = (USHORT)WinSendMsg(hwndScroll, SBM_QUERYPOS, 0L, 0L);
- Set the position with
  - –WinSendMsg(hwndScroll, SBM_SETPOS, . . .)
- Scrollbars understand WM_CHAR messages (PgUp, PgDn)

Scrollbar Notification Messages

- WM_VSCROLL, WM_HSCROLL
  - –id = SHORT1FROMMP(mp1) /* scrollbar ID */
  - –sPos = SHORT1FROMMP(mp2)
  - –usCmd = SHORT2FROMMP(mp2)
    - ►*SB_LINEUP* *SB_LINELEFT*
    - ►*SB_LINEDOWN* *SB_LINERIGHT*
    - ►*SB_PAGEUP* *SB_PAGELEFT*
    - ►*SB_PAGEDOWN* *SB_PAGERIGHT*
    - ►*SB_SLIDERPOSITION*
    - ►*SB_SLIDERTRACK*
    - ►*SB_ENDSCROLL*

Entry-Field Controls

- Typically used in dialog windows
- Displays a single line of text which a user can edit.
- When the control has the focus, it displays a flashing insertion-point cursor
- Hold up to 32 characters by default
- Override defaults by passing an entry-field creation structure with appropriate parameters
- Major message from entryfields: WM_CONTROL with notification codes
  - EN_SETFOCUS
  - EN_KILLFOCUS
  - EN_CHANGED

Multiple Line Entry (MLE) Fields

- Provide an editor in a window
- Automatic word-wrap, tab stops, line and character counting
- Automatic search and undo
- Full font support
- MLE Styles:
  - MLS_BORDER — Draws a border round the MLE
  - MLS_HSCROLL — Adds a horizontal scroll bar
  - MLS_IGNORETAB — Ignores the TAB key
  - MLS_READONLY — Won't accept text from the user
  - MLS_VSCROLL — Adds a vertical scroll bar
  - MLS_WORDWRAP — Breaks lines automatically
- Expensive (250 KB!)

Messages sent to an MLE

- MLM_COPY
  - –Copies selected text to the clipboard
- MLM_CUT
  - –Cuts selected text to the clipboard
- MLM_EXPORT
  - –Exports text from the MLE (e.g. for file save)
- MLM_IMPORT
  - –Imports text into the MLE (e.g. for file read)
- MLM_INSERT
  - –Inserts text into the MLE
- MLM_PASTE
  - –Copies the clipboard contents to the MLE
- MLM_QUERY?????
  - –Queries various MLE settings

Messages Sent by an MLE

- WM_CONTROL, with notification codes:
- MLN_CHANGE
  - Text in the MLE has changed
- MLN_HSCROLL, MLN_VSCROLL
  - Horizontal and vertical scroll events
- MLN_KILLFOCUS
  - MLE has lost the input focus
- MLN_SETFOCUS
  - MLE received input focus
- MLN_TEXTOVERFLOW
  - MLE text-limit overflow
- MLN_UNDOOVERFLOW
  - Indicates text change cannot be undone

WC_LISTBOX

- Used to display a list in a scrolling window and allow the user to make a selection
- Limits: 32,767 items, 64K heap-size limit
- List Styles
- LS_MULTIPLESEL
  - Allow more than one selection at a time
- LS_NOADJUSTPOS
  - Does NOT make list-box height a multiple of the item height to avoid displaying a partial item at the bottom
- LS_OWNERDRAW
  - Causes the owner window to receive a WM_DRAWITEM message each time an item must be drawn or highlighted

Messages Sent to a List Box

- LM_INSERTITEM
  - Inserts an item in the list box, at a specified position, the beginning or end of the list, or in ascending or descending order
- LM_QUERYITEMCOUNT
  - Returns the number of items in the listbox
- LM_QUERYSELECTION
  - Returns the index of the selected item
- LM_QUERYITEMTEXT
  - Copies the text of the specified item into a buffer
- LM_SELECTITEM
  - Sets the selection state for the specified item

Messages Sent by a List Box

- WM_CONTROL, with notification codes:
- LN_ENTER
  - User pressed enter or double-clicked
- LN_SELECT
  - User selected an item
- LN_SCROLL
  - List box scrolled

Combo Boxes

- A combo box combines an entry-field with a list box, and automatically manages interaction between the two controls
- Combo Box Styles:
  - CBS_SIMPLE
    - *Always displays its list box*
  - CBS_DROPDOWN
    - *Displays a list box if the user clicks the drop-down icon at the right end of the entry field*
  - CBS_DROPDOWNLIST
    - *User can only select from the list and not enter text*
- Messages are similar to those for a list box

Buttons

- Buttons are different from other control windows in that, like menus, they send WM_COMMAND messages
- Types of button:
  - BS_PUSHBUTTON
  - BS_HELP (help button - posts WM_HELP)
  - BS_CHECKBOX
  - BS_3STATE
  - BS_AUTO3STATE
  - BS_RADIOBUTTON (mutually exclusive selection)

Day 2 – Session 2

Lab Exercise 2 – Menus and Messages

Day 2 – Session 3

Memory Management

OS/2 Kernel Features

Memory Management

Kernel Features

- Support for:
  - Multitasking
  - Virtual Memory
  - Firewalls between processes
  - Installable Components
  - Dynamic linking of segments
  - Device drivers
  - Inter-process communications

Applications Programming Interface

- The API is the way in which applications gain access to system services
- The DOS API consists of:
  - Place function number in AH (or AX if subfunction also)
  - Parameters in DX or ES:BX
  - Execute INT 21H
- The API consists of the code, plus supporting documentation and tools

The OS/2 API

- The OS/2 API is based on the CALL instruction
  - Because there are no software interrupts in protected mode
  - The technique is portable between real and protected modes (though DOS doesn't support CALLs)
  - Directly callable from High Level Languages
  - Easily dynamically linked
- All parameters are passed on the stack
- A result code is returned in AX
- OS/2 1.x: Pascal calling convention:
  - 16-bit parameters are pushed from left to right
  - Called function removes all parameters from the stack
- OS/2 2.x: _System (C) calling convention:
  - 32-bit parameters pushed from right to left
  - Calling function removes all parameters from the stack

Sessions (Screen Groups)

- A Session (or Screen Group, in 1.x terminology - the header files still contain SG_* constants) is a virtualized:
  - Screen
  - Keyboard
  - Rodentiometer
- Presentation Manager occupies a single screen group
- Each full-screen DOS session (the DOS compatibility box in OS/2 1.3) occupies another
- Each full-screen character/kernel application occupies another

OS/2 I/O Services

- Video I/O
  - Modeled on ROM BIOS INT 10H
  - Implemented as a DLL - replaceable
  - VIO calls not supported for 32-bit apps in OS/2 2.x
    - *Get around this by thunking*
    - *Text-mode Vio will be officially supported soon*
- Keyboard I/O
  - Follows ROM BIOS INT 16H
  - Implemented as a DLL
- Mouse
  - Based on MS Mouse INT 33H
  - Implemented as a DLL
- Character Device Monitors
  - Allow apps to intercept character screens
  - Replace TSR s/w, e.g. keyboard redefinition, macro expansion
  - 16-bit only under 2.X

OS/2 Kernel / Character Applications

- Run in protected mode only
- May run in a window or own screen group
- But. . . cannot utilise Presentation Manager features
- Must utilise 16-bit VIO subset of API or ANSI escape sequences for screen manipulation
- Typically character only, such as
  - Applications with either very simple or no user interface
    - *Compilers*
    - *Linkers*
    - *Sort*
    - *UNIX-style pipes*
  - Time-critical applications
  - Daemon processes
    - *Print spoolers*
    - *Network services*

Daemon / Detached Processes

- Can be started from the command line
  - DETACH <appname>
- or from CONFIG.SYS
  - RUN = <appname>
- A detached process does not appear on the application selector list and runs in an unselectable screen group
- Must use device monitor input for keyboard / mouse
- Must use VioPopup for screen output
- Must provide own interface for termination
- Must provide hard error handler

C Compiler Memory Management Functions

- void *malloc(size_t size)
  - "Buys in bulk, sells in small quantities, cheaply"
  - malloc returns a near/far pointer to a block of at least size bytes from the default data segment
- size_t _memavl(void) - 16-bit only
  - Returns the approximate amount of memory available for dynamic memory allocation in the near heap (default data segment)
- void *calloc(size_t num, size_t size)
  - returns a near/far pointer to space for an array of num elements of size bytes. All bytes are initialized to 0.
- void *realloc(void *memblock, size_t size)
  - Changes the size and possibly the location of the block
- void free(void *memblock)
  - Frees a memory block previously allocated by calloc, malloc, or realloc
- int _heapmin()

16-bit (OS/2 1.X) Memory Management

- Allows overcommitment of physical memory
- Virtual Segmentation allows
  - Larger programs than physical memory
  - More programs than physical memory
- Segments are moveable
- Inactive segments can be swapped
- Segments may be discarded in preference to swapping (and are, in OS/2 1.3)

OS/2 1.X Swapping

- Is performed on a per-segment, not per-application, basis
- Applications cannot see segment swapping
- The segment size can and should be tuned using separate compilation and SEGMENT entries in the .DEF file
- Loading is controlled by .EXE file advisories (flags in the header)
  - Segment preload
  - Load on demand
  - Load on call
  - Controlled by a .DEF file
- LRU swapout algorithm with memory commitment tracking
  - Fixed Memory - kernel code/data, swapping management, interrupt handlers
  - Deadlock prevention memory

16-bit Memory Allocation

- In segmented (286) model, DosAllocSeg function
- DosAllocSeg is not supported or required in OS/2 2.x
- Shared Segments
  - Give-away shared segments
  - Named shared memory
    - *In file system namespace, allowing*
    - *Permissions (future)*
    - *Networking (future)*
- Huge segments
  - Multiple 64K segments, based on a base segment and segment spacing
- CSAlias
  - Technique to provide both data and code selectors for a single segment

OS/2 1.x Memory Management Functions

- To allocate a segment and create a pointer to it:
  - SEL sel;
  - PCH pch;

  - DosAllocSeg(size, &sel, SEG_NONSHARED);
  - pch = MAKEP(sel, 0);
- To resize the segment:
  - DosReallocSeg(new_size, sel);
- To free the segment:
  - DosFreeSeg(sel);
- To allocate a sequence of contiguous selectors (huge segment)
  - DosAllocHuge(no_segs, last_bytes, &selHuge, no_res_sels, SEG_NONSHARED);

OS/2 1.X Allocation Flags

- SEG_DISCARDABLE
  - Segment may be discarded
- SEG_GETTABLE
  - Segment is shareable - another process can retrieve it using the DosGetSeg function
- SEG_GIVEABLE
  - Segment is shareable - can be given to other processes using the DosGiveSeg function
- SEG_NONSHARED (default)
  - Segment is non-shareable and nondiscardable

Simple Heaps

- OS/2 provides two methods of creating heaps
  - Heap Manager
  - Simpler technique shown here
- To create an 8 KB heap:
  - SEL selHeap;

  - DosAllocSeg(8192, &selHeap, SEG_NONSHARED);
  - DosSubSet(selHeap, 1, 1024);
- To suballocate:
  - USHORT offBlock;
  - PBYTE pb;

  - DosSubAlloc(selHeap, &offBlock, 1024); /* 1KB in block */
  - pb = MAKEP(selHeap, offBlock);
  - .
  - DosSubFree(selHeap, offBlock, 1024);

More On Heaps

- OS/2 reserves 12 bytes in each heap for its own use
- DosSubAlloc always rounds up to a multiple of 4 bytes
- Take care when using pointers to memory blocks. OS/2 provides no protection against accidental misuse!
- Heaps can be resized by calling DosReallocSeg and DosSubSet again.
- The entire heap is removed by the DosFreeSeg function call
- The HEAPSIZE parameter in .DEF files has no bearing on heaps allocated within application-allocated segments.

Heap Manager

- More functionality than basic memory-management functions
- Faster allocation implementation
- Moveable objects within a segment
- Created with
- HHEAP hHeap;
- hHeap = WinCreateHeap(selHeapBase, /* Heap Selector */
- cbHeap, /* Initial size */
- cbGrow, /* Increment by */
- cbMinDed,
- cbMaxDed,
- fsOptions);
- selHeapBase == 0 => Heap in automatic data segment
- cbHeap == 0 => Heap size set from .DEF file
- cbMinDed, cbMaxDed used to set up dedicated free lists for optional faster operation.

Memory Management

- 1.x:
  - Segmented model has segments of 1B - 64 KB
  - Segment base + offset combination makes pointer arithmetic painful
  - Complexities of memory models: Small, compact, medium, large and huge
  - Swapping is degraded by fragmentation of the swap file, allowing only nominal overcommitment of memory

Memory Management (cont)

- The 386 processor has a paging mechanism, based on page translation tables
- A 32-bit address actually consists of:
  - 10-bit page directory
  - 10-bit page
  - 12-bit offset within a 4 KB page
- Segmentation still takes place, but now using a 32-bit segment selector and 32-bit offset
- In OS/2 2.x, the application programmer only deals with offsets
- This is called the 0:32 model

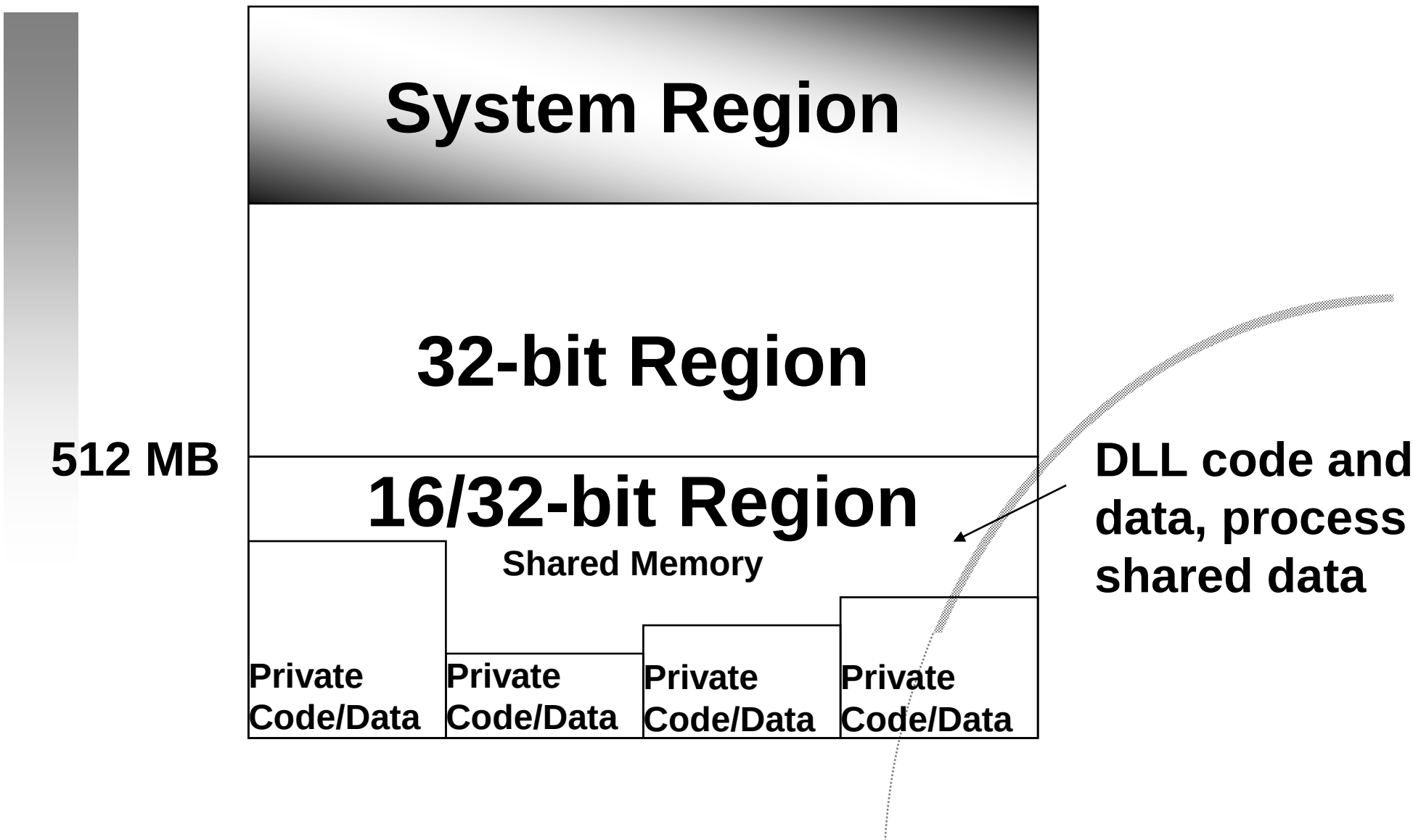
Memory Management (cont)

- In OS/2 2.X, a range of memory allocated to a process is called a memory object
- A memory object can be up to 512 MB in size
- Allocation of an object actually reserves the required number of pages
- Therefore the allocated memory is rounded up to a multiple of the page size
- Attempts to access beyond the end of an object, but within the last page allocated for it, will not cause an error
- Do not assume that the page size is always 4KB!
  - Use DosQuerySysInfo(QSV_PAGE_SIZE, QSV_PAGE_SIZE, &buffer, buflen);

Memory Management (cont)

- Memory is not relocatable or resizable, so allocation is done by
  - 1. Allocating the memory but not committing it. This allocates virtual memory but not physical
  - 2. Committing the memory object (or part of it) to physical memory.
- Note the very important distinction between allocation and commitment
- Allocate more than you expect to need, then commit what is required

Process Address Space



OS/2 2.x Memory Management Functions

- To allocate a memory object and create a pointer to it:
  - PVOID pv;
  - ULONG allocflags;

  - allocflags = PAG_COMMIT | PAG_READ | PAG_WRITE;
  - DosAllocMem(&pv,size, allocflags);
- To resize the memory object:
  - Not possible in OS/2 2.0!
- To change allocation attributes:
  - DosSetMem(pv, region_size, allocflags);
- To query attributes on a memory object:
  - DosQueryMem(pv, &region_size, &allocflags);
- To free the memory:
  - DosFreeMem(pv);
- Remember to check return codes!

OS/2 2.0 Allocation Flags

- PAG_COMMIT
  - All pages in the private memory object are initially committed
- OBJ_TILED
  - Object must be allocated in the first 512 MB of virtual-address space, with 16-bit selectors for compatibility
- PAG_READ
- PAG_WRITE
- PAG_EXECUTE
- PAG_GUARD
  - Page is a guard page and access will trigger an exception

OS/2 2.1 Heap Functions

- To prepare a memory object for suballocation:
  - PVOID pheap, pblock; ULONG subflags, size, heapsize;
  - allocflags = PAG_READ | PAG_WRITE | PAG_COMMIT;
  - subflags = DOSSUB_INIT;
  - DosAllocMem(&pheap, heapsize, allocflags);
  - DosSubSetMem(pheap, subflags, heapsize);
- To allocate a block of memory from the pool:
  - DosSubAllocMem(pheap, &pblock, size);
- To free a block of memory from the pool:
  - DosSubFreeMem(pheap, pblock, size);
- To end use of the memory pool:
  - DosSubUnsetMem(pheap);

Suballocation Flags

- DOSSUB_INIT
  - Must be set to initialize a memory object for suballocation. Otherwise, attaches a process to another process's memory pool
- DOSSUB_GROW
  - Request is to increase the size of the memory pool.
- DOSSUB_SPARSE_OBJ
  - Causes the suballocation functions to manage commitment of the pages of the pool
- DOSSUB_SERIALIZE
  - Causes access to the heap to be serialised

LDT Tiling

- Shared process region grows from top down
- Private process region expands upwards in shared memory
- This pattern is called LDT tiling and replaces the disjoint LDT space approach used in OS/2 1.x

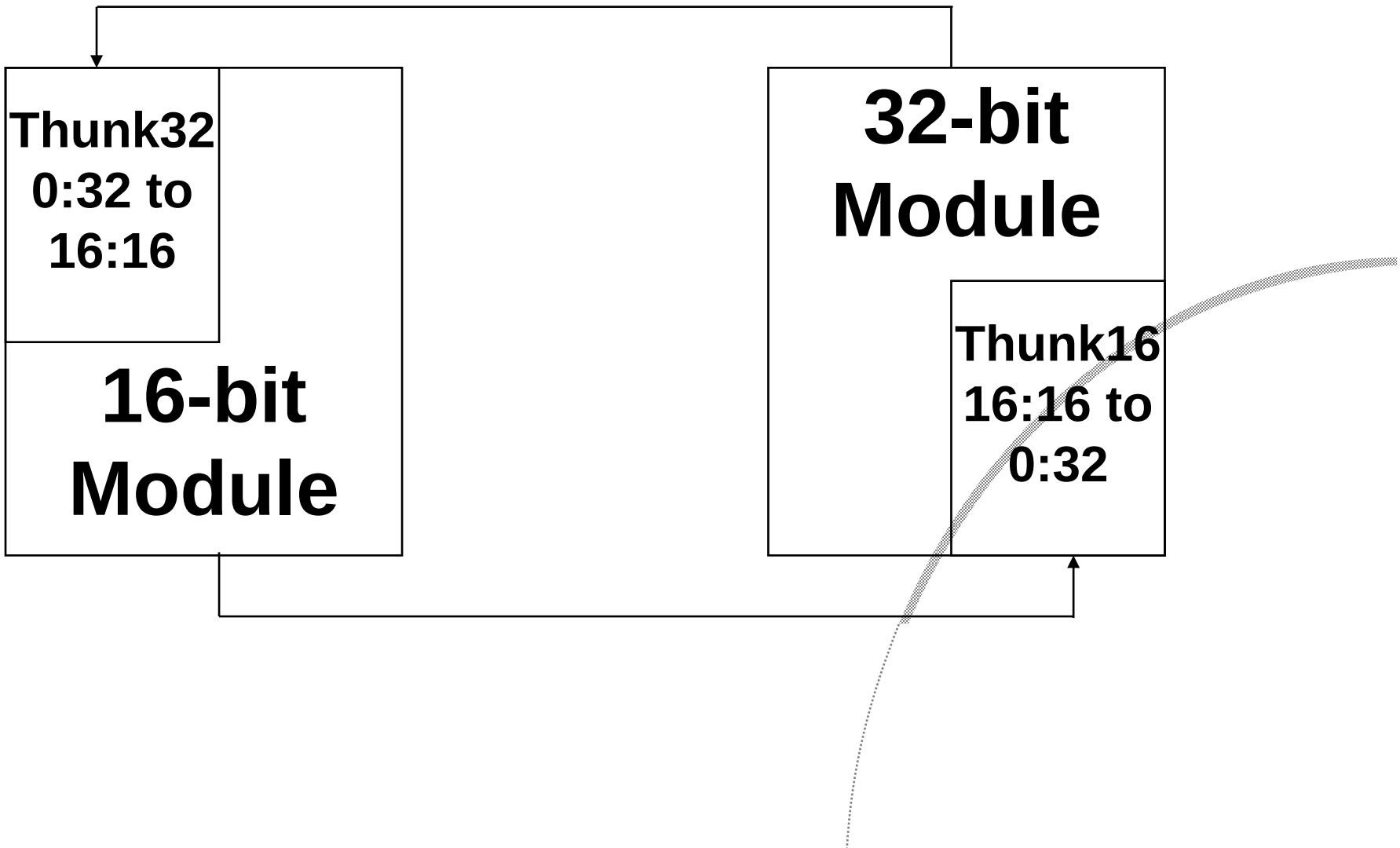
Mixed Environment

- OS/2 2.x supports both OS/2 1.x (16:16) and OS/2 2.x (0:32) applications.
- Problems:
  - Running 16-bit applications in a 0:32 environment
  - 0:32 applications calling 16:16 DLL's
  - 16:16 applications calling 0:32 DLL's
  - Large memory objects (over 64 KB)
- These problems are resolved by using thunks

Thunks

- Thunks are routines which translate between:
  - The address model used (16:16 vs 0:32)
  - Different parameter sizes
    - *OS/2 1.x : 16-bit (SHORT, WORD) Parameters*
    - *OS/2 2.x : 32-bit (LONG, DWORD) Parameters*
  - Stack-based addressing (WORD vs DWORD)
  - Large objects (>64KB) passed to 16:16 code
  - Different call models
    - *OS/2 1.x : API calls are far calls*
    - *OS/2 2.x : API calls are near calls*

Thunks (cont)


Thunk32
0:32 to
16:16
16-bit
Module
32-bit
Module
Thunk16
16:16 to
0:32

Day 2 – Session 3

Lab Exercise 4 – Memory Management

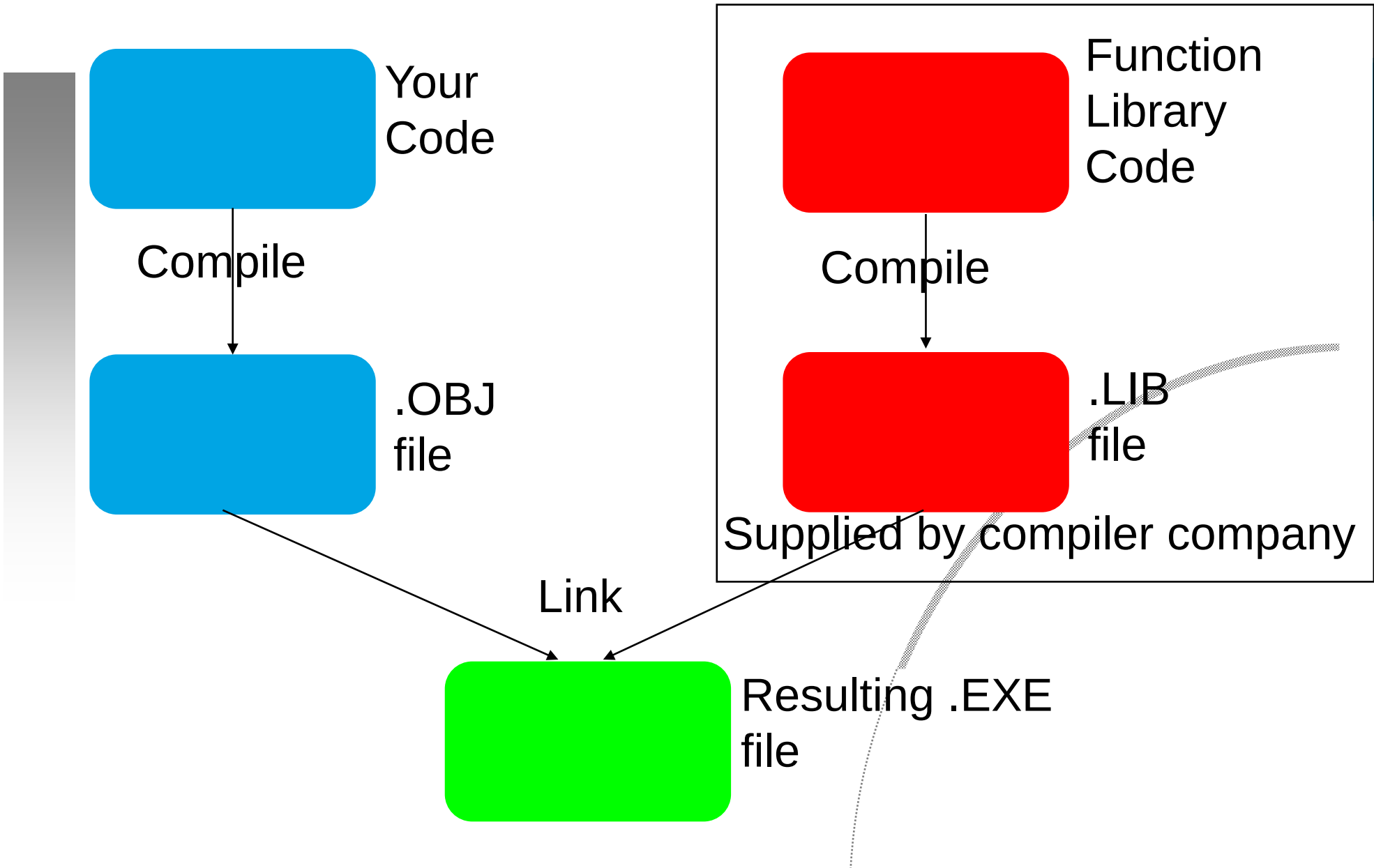
Day 2 – Session 4

Dynamic Link Libraries

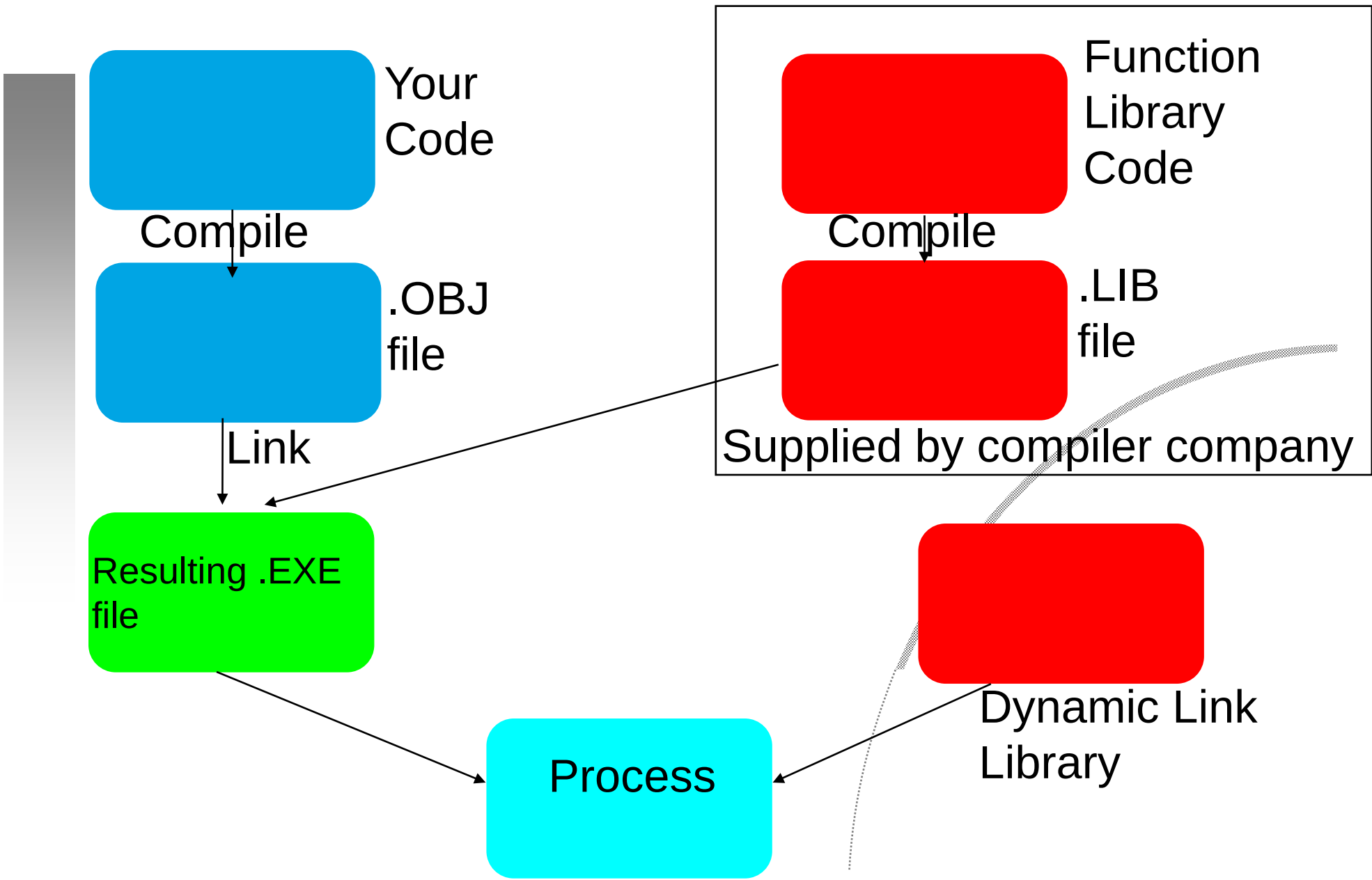
Dynamic Link Libraries

Subtle, but important

The Linking Process



Dynamic Linking



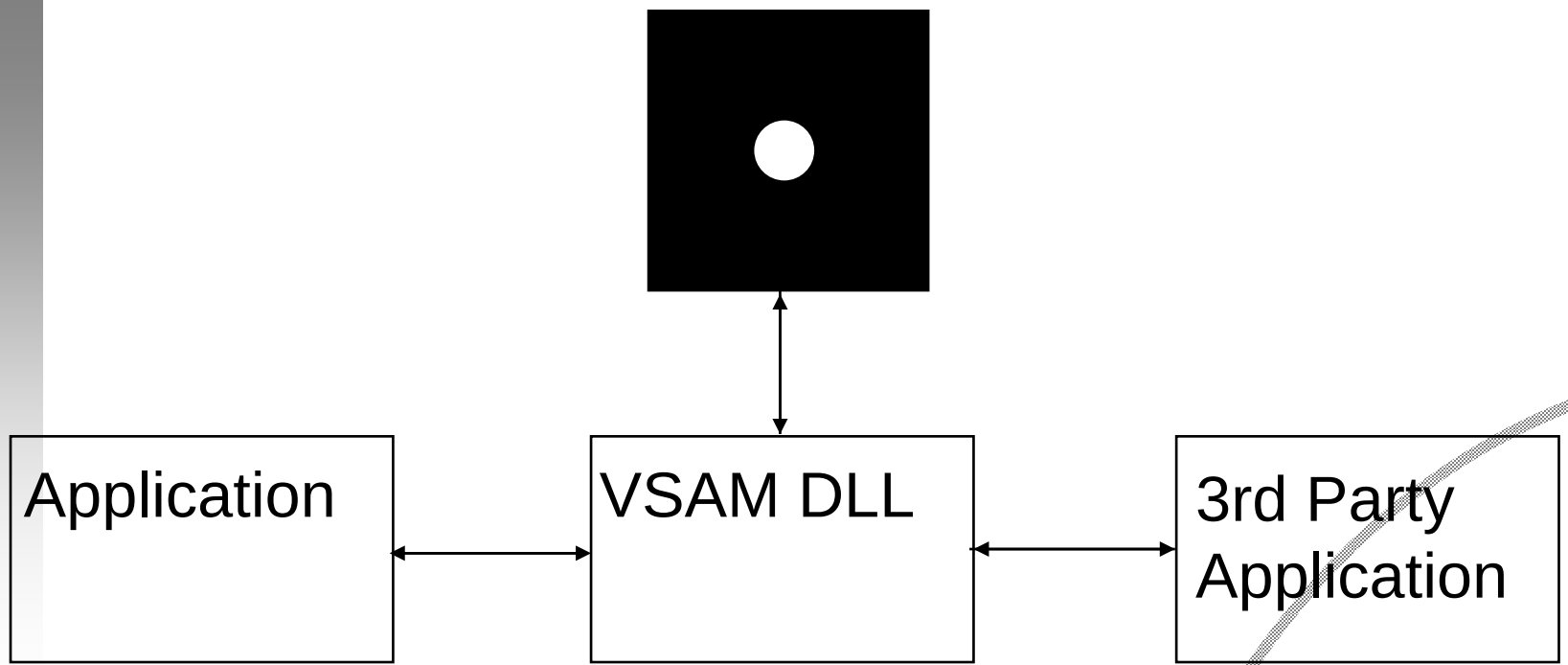
Dynamic Linking

- is delayed binding of the application's external references to subroutines until either
  - Load time, or
  - Run time
- All system function requests are made via dynamic linking
- Applications may comprise .EXE files and .DLL files

Benefits of Dynamic Linking

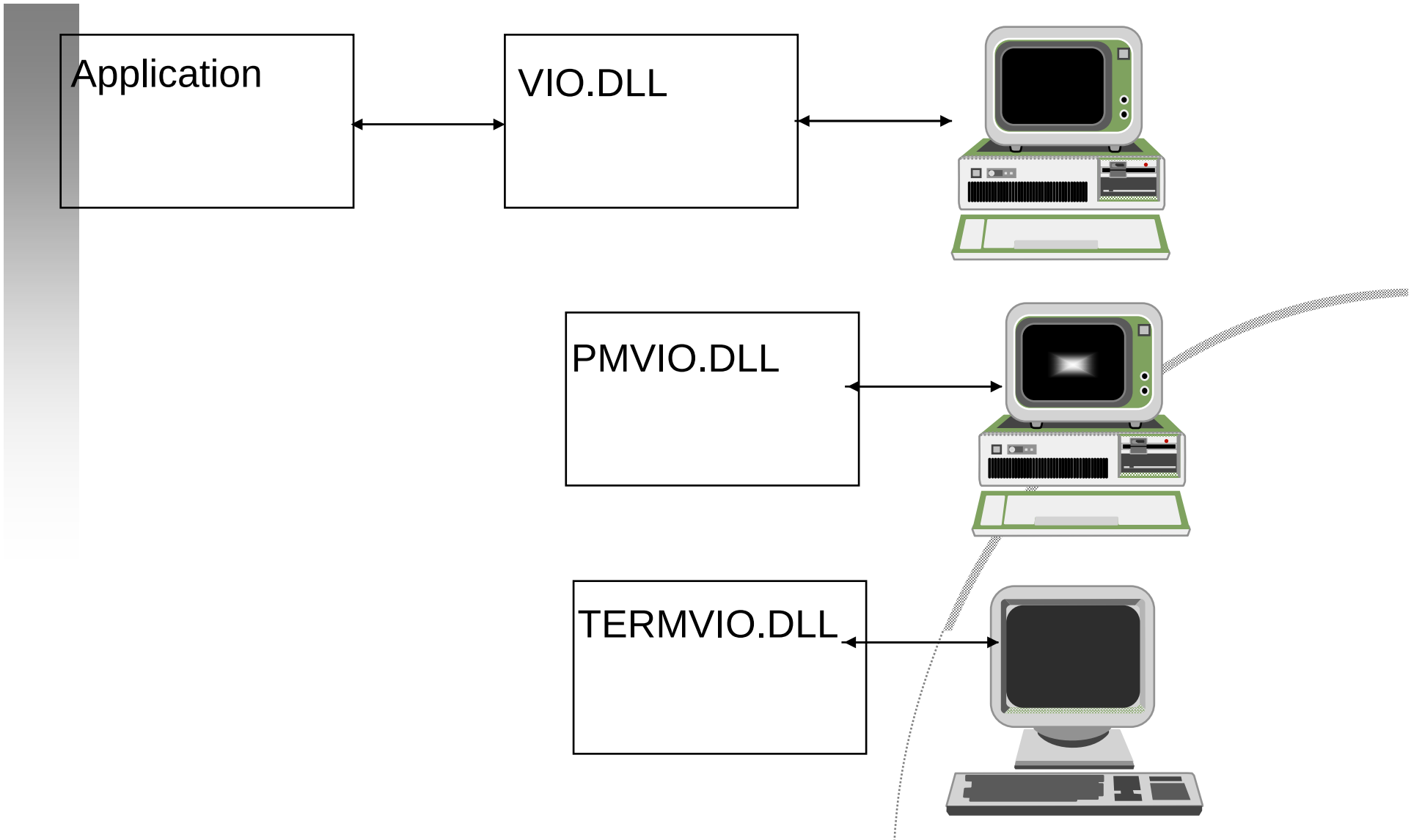
- OS/2 API is extensible - provide your own dynlink libraries
- Shareable code segments
- Smaller .EXE files
- Faster loading for multiple invocations
- Demand loading
- Down-side:
  - Slower initial load

DLL's and Object-Oriented Design

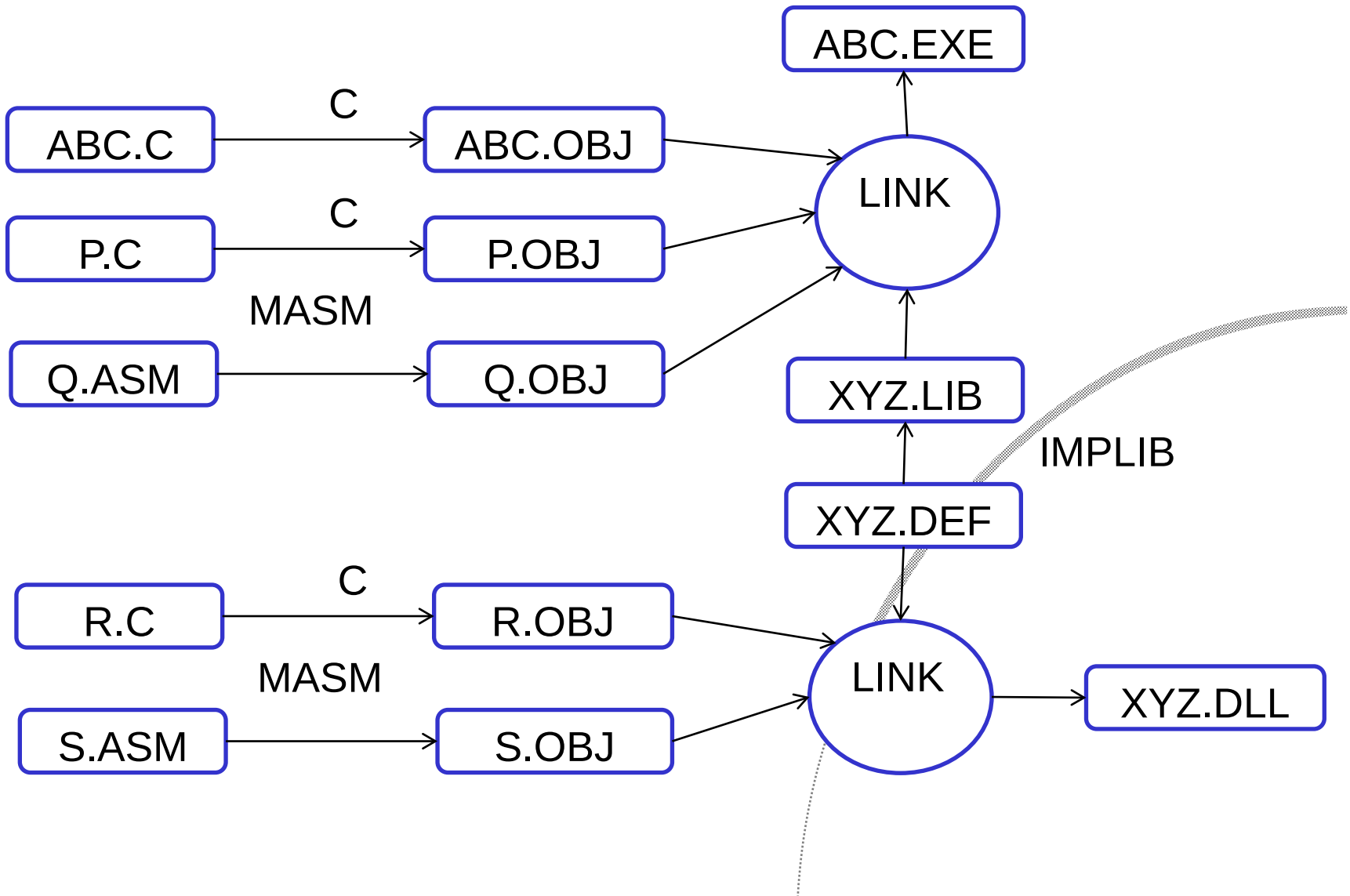


- Third party applications can operate with your files
- Any file format changes will be invisible - just the DLL changes

DLL's Allow Apps to be Device-Independent



Creating and Using a DLL



Creating a DLL

- Create a .DEF file for the library which EXPORTS the required functions
- Declare all exported functions EXPENTRY
- Do not use the _Export declarator in CSet++
- Either:
  - create a .DEF file for the main application files which IMPORTS the required functions, or
  - use IMPLIB to 'compile' the DLL's .DEF file to create an import library
- For 16-bit code only:
  - If your DLL uses static data, each exported function will require the DS register to be reloaded on entry. Use the _loadds keyword to do this.
  - If the DLL does not use static data, do not do this.

Checklist for DLL Creation

- To build a DLL:
  - Use compiler option /Ge-
  - EXPENTRY keyword on exported functions
  - .DEF file
    - *LIBRARY xyz*
    - *EXPORTS*
    - *func1 @1*
    - *func2 @2*
  - Specify .DEF file in link options
  - Use IMPLIB to build an import library (WF does this for you)
- To build a client program:
  - LINK with the import library

Types of Dynamic Linking

- Load-time Dynamic Linking
  - Benefits:
    - *Simplest form of dynamic linking*
    - *System automatically loads and links DLL's*
  - Drawbacks:
    - *Failure to load a DLL will terminate loading*
    - *DLL's must be located on the LIBPATH*
- Run-time Dynamic Linking
  - Benefits
    - *Can work out DLL names at run-time*
    - *Can use DLL's to support multiple subsystems*
    - *Application retains control after a DLL fails to load and can continue or terminate*
    - *DLL's can be loaded from anywhere*
  - Drawbacks
    - *More complex to program*

DLL-Related Functions

- Interesting functions related to DLL's:
- DosLoadModule
  - Loads the specified DLL and sets a module handle. WinCreateWindow can now refer to resources in this DLL by their handle
- DosQueryModuleName
  - Retrieves the name and path of a module from a module handle
- DosQueryProcAddr
  - Retrieves the address of a specified function in a DLL module, so that it can be called.
- DosFreeModule
  - Frees the module.
- DosQueryModuleHandle
  - Retrieves the module handle for the specified DLL.
- DosSetExtLIBPATH (Warp)
  - Defines path to be searched before or after LIBPATH

Day 2 – Session 4

Lab Exercise 5 – Dynamic Link Libraries

Agenda

- Day 1
  - Session 1 – Introduction to Tools
  - Session 2 – Introduction to PM
  - Session 3 – Lab Exercise 1
  - Session 4 – Windows Parentage and Ownership
- Day 2
  - Session 1 – Window Controls
  - Session 2 – Lab Exercise 2 – Menus and Messages
  - Session 3 – Memory Management
  - Session 3 – Lab Exercise 4 – Memory Management
  - Session 4 – Dynamic Link Libraries
  - Session 4 – Lab Exercise 5 – Dynamic Link Libraries

Agenda

- Day 3
  - Session 1 – Threads, IPC and File I/O
  - Session 2 – Lab Exercise 6 - Threads
  - Session 3 - Workshop
  - Session 4 – Filesystems % EA's
  - Session 4 – Lab Exercise 8 – Directory Listing
- Day 4
  - Session 1 – Window Words, Subclassing, Dialogs
  - Session 2 – Lab Exercise 9 – Multiple Windows and Instance Data
  - Session 3 – Lab Exercise 9 continues
  - Session 4 – Standard Dialogs and INI files
- Day 5
  - Session 1 – Graphics Programming Interfase
  - Session 2 - Workshop
  - Session 3 – SOM and WPS
  - Session 4 – It's Friday…

Day 3 – Session 1

Threads, IPC and File I/O

Threads - Or How To Walk Down the Street and Chew Gum At The Same Time

OS/2 Multitasking

Multitasking Concepts

- A *program* is a set of instructions on disk
- When a program is loaded and run, it becomes a *process*
- Simplest multitasking: load the same program twice to create two processes:
  - The two processes have identical code segments, wasting space, so
- Better multitasking:
  - Two processes can share code segments
  - But must obviously have their own data segments
  - When we kill a process, we must not always delete the code segments.

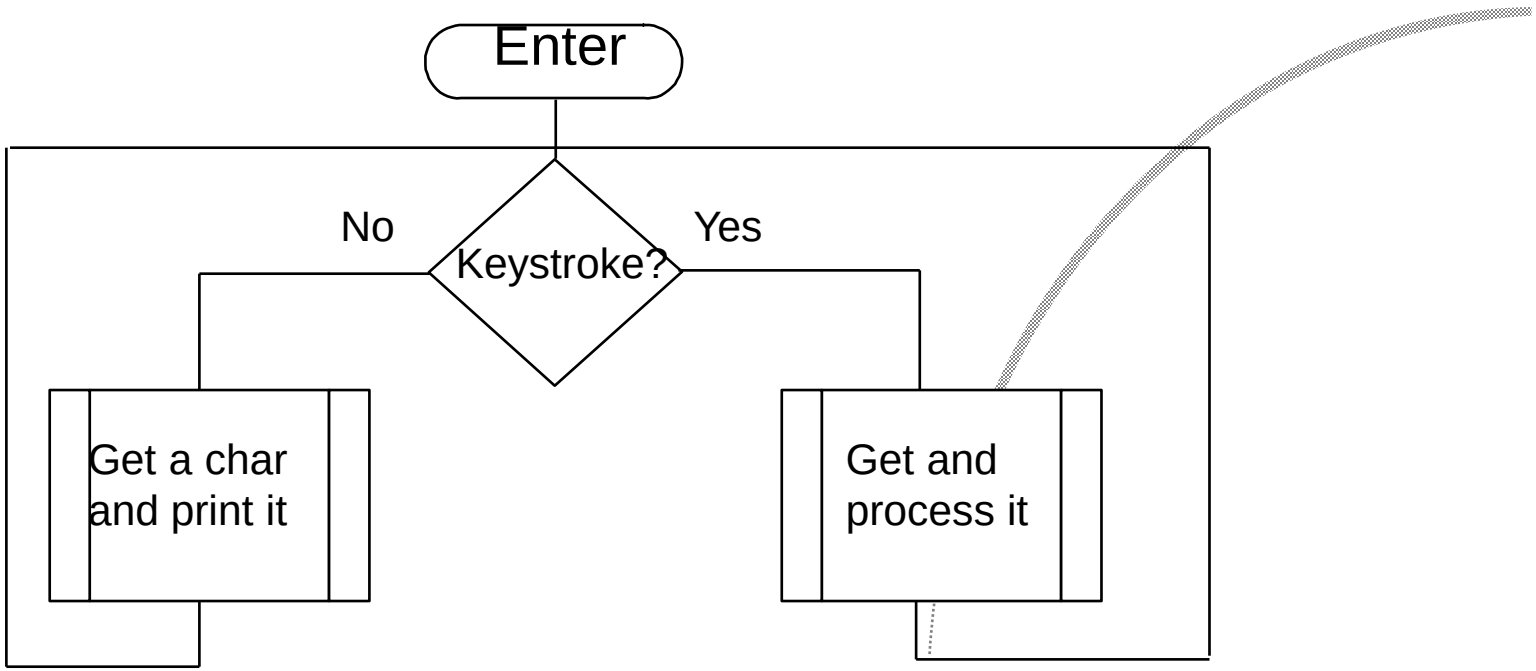
Starting a Child Process

- Operation can be synchronous (EXEC_SYNC) or asynchronous (EXEC_ASYNC, EXEC_ASYNCRESULT)
- The termination code and result code can be examined later using the DosWaitChild() API

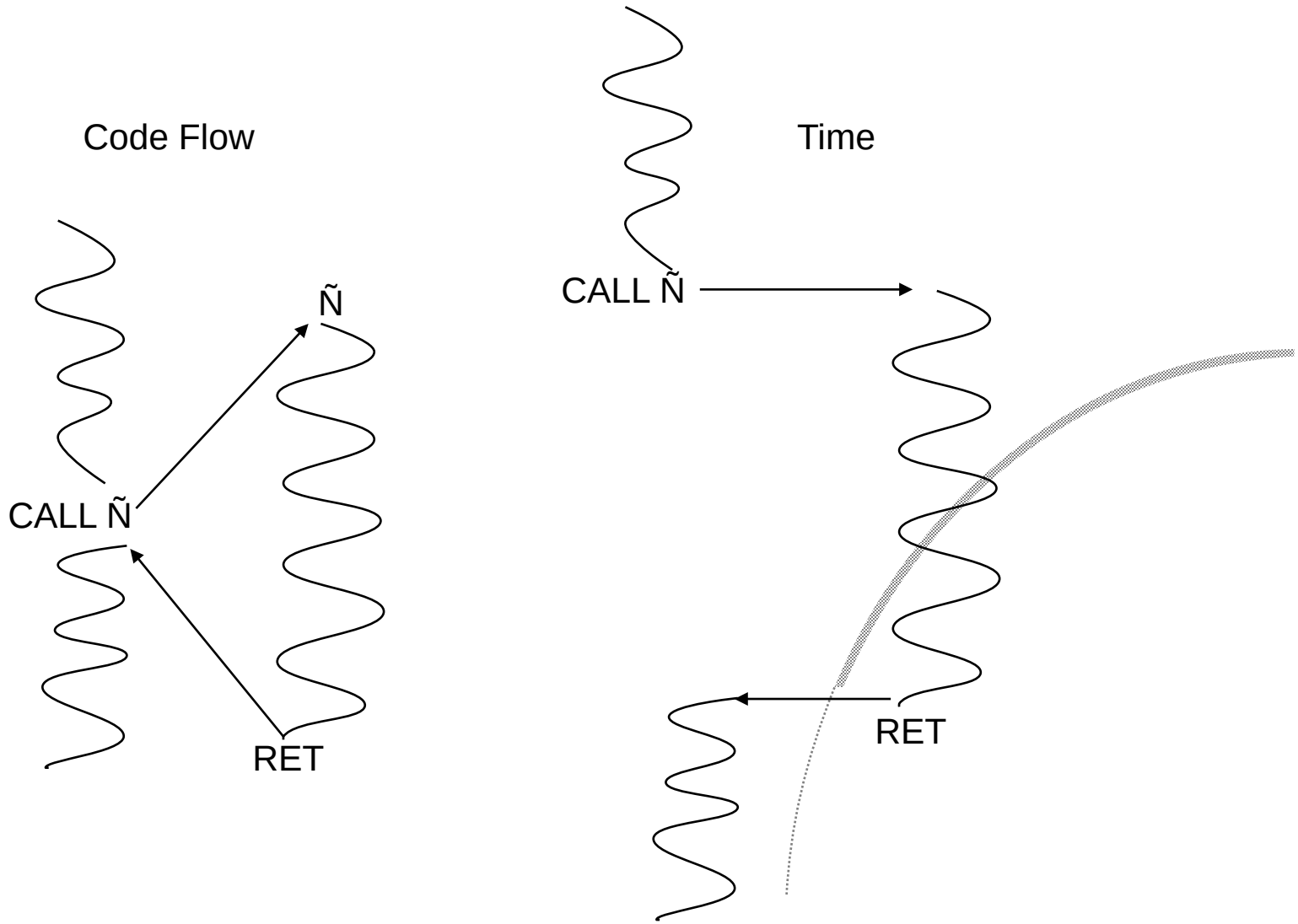
```
PCHAR          ObjNameBuf;
LONG           ObjNameBufL;
ULONG          ExecFlags;
PSZ            ArgPointer;
PSZ            EnvPointer;
PRESULTCODES   ReturnCodes;
PSZ            PgmPointer;
APIRET         rc;    /*  Return Code. */
rc = DosExecPgm ( ObjNameBuf,  ObjNameBufL,
                  ExecFlags,  ArgPointer,  EnvPointer,
                  ReturnCodes,  PgmPointer);
```

Internal Multitasking

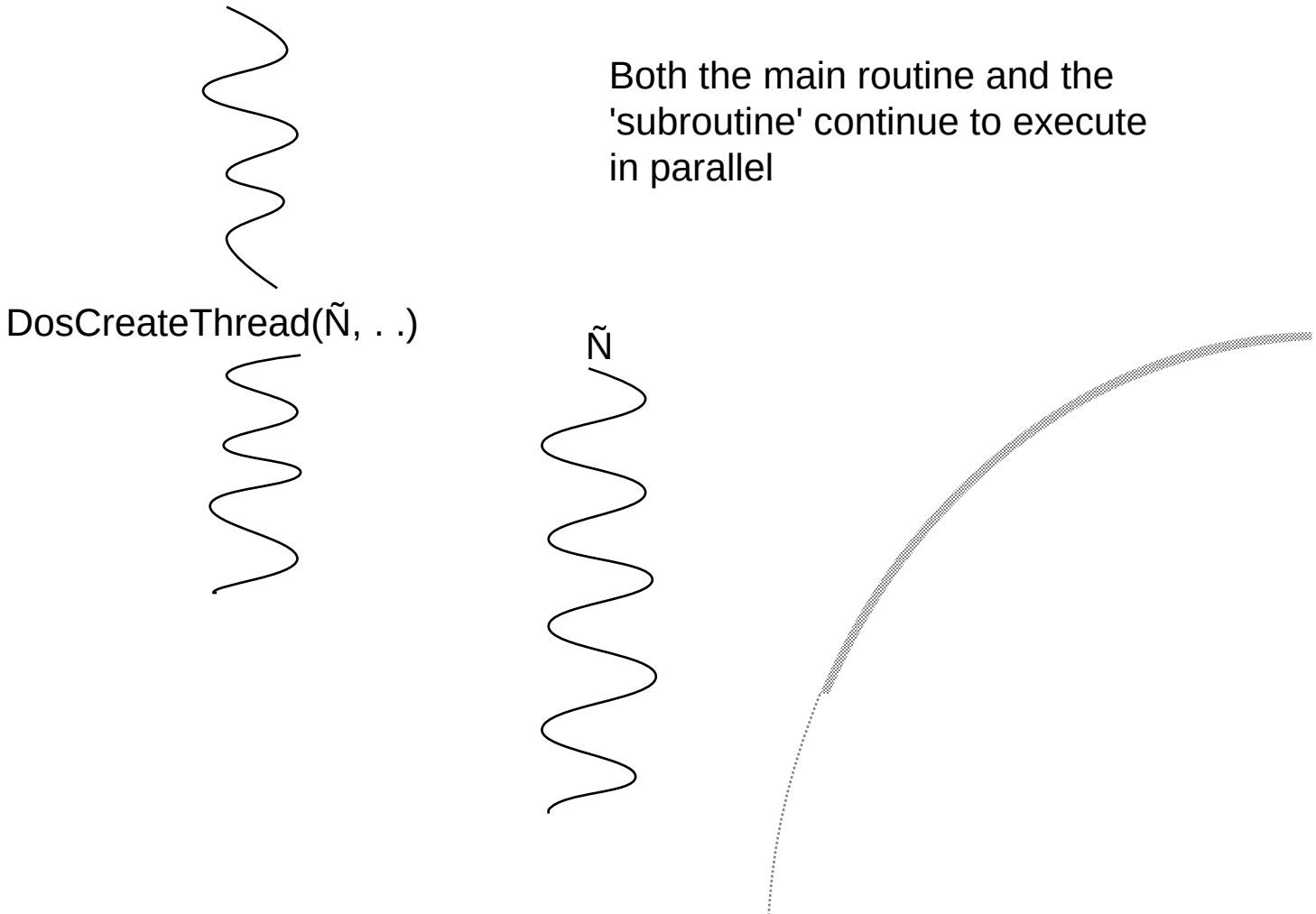
- Many programs do two things at once, e.g.
  - Even CP/M WordStar could print and edit simultaneously
  - DOS spreadsheets which recalc in background
- This is done by having a loop which checks for keyboard input, and if none, does something else



Flow of Execution - Subroutines



Flow of execution - Threads



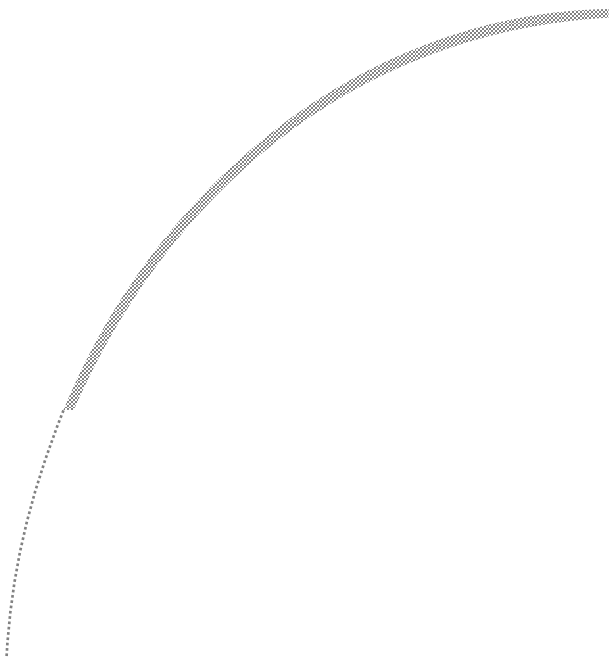
Both the main routine and the 'subroutine' continue to execute in parallel

Processes vs Threads

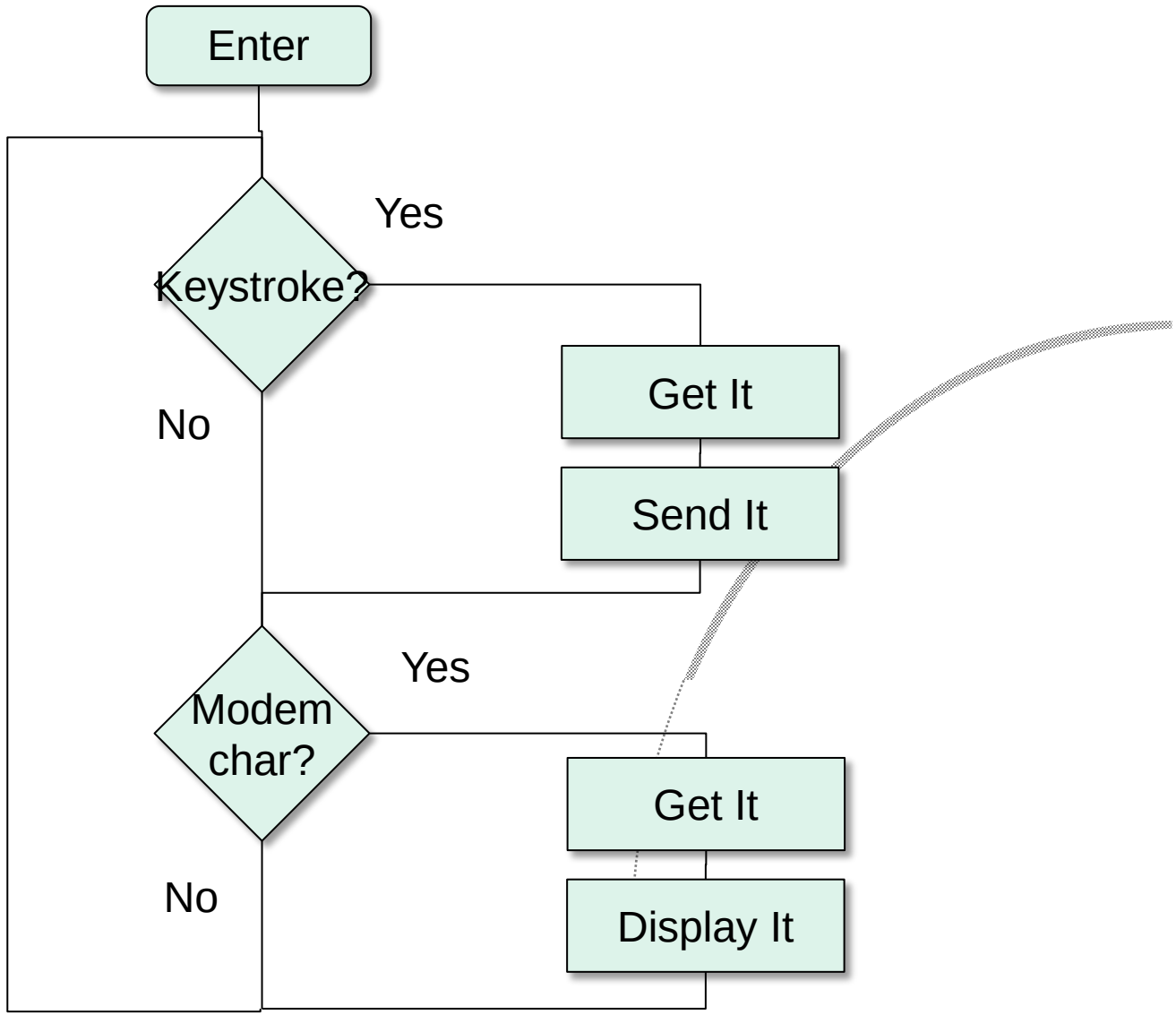
- A process is the instance of program execution. It is the OS/2 unit of resource ownership.
- A thread is a dispatchable entity. It is the OS/2 unit of execution.
- The scheduler dispatches threads, not processes.
- Processes own things
- A thread owns two things:
  - Its priority
  - Its registers
- An OS/2 process can, and often should, comprise multiple threads, all running simultaneously.

Applications for Threads

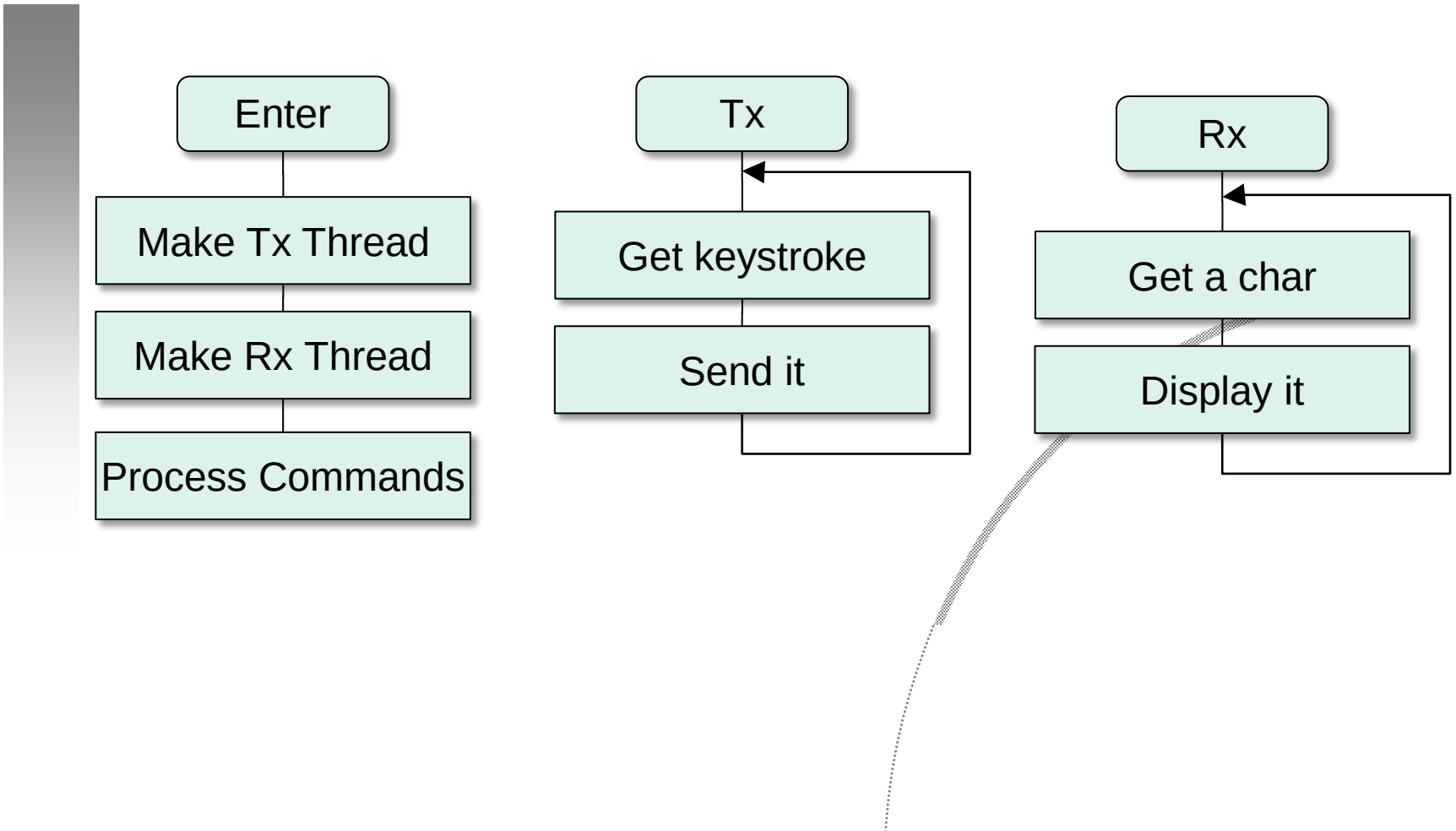
- Simultaneous printing
- Background recalculation
- Performance monitoring
- Time-consuming background activities
- Making the system more responsive
- Simplifying designs



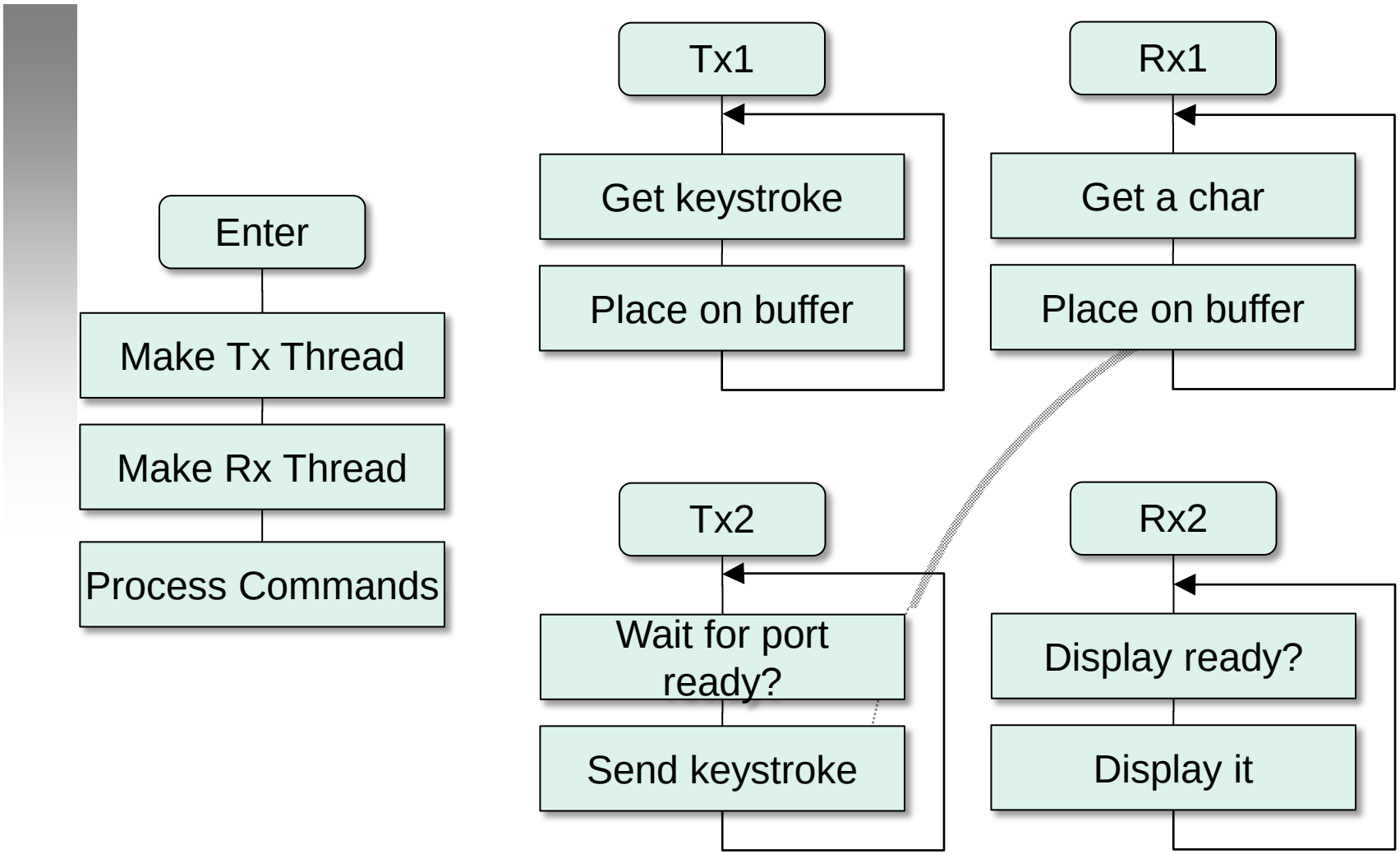
Tradition Terminal Program Design



OS/2 Terminal Program Design



OS/2 Double-buffered Terminal Program Design



OS/2 Task Scheduler

- Preemptive
- Timeslicing (32 ticks per second)
- Schedules threads by priority
- Four classes, each with 32 priority levels
  - Time Critical Class
    - *Highest class*
    - *Round robin within level*
  - Fixed-High Priority (Server) Class
  - Regular Class
    - *Priority varies based on foreground/background and I/O vs CPU usage*
    - *Can limit time denied the CPU*
  - Idle Time Class
    - *Lowest priority class*

Why Threads Are Important

- Presentation Manager is a message-passing system
- There are multiple queues, but their operation is synchronous
  - The system will not read the next message until a winproc has returned from processing the previous one
- If one winproc holds up the thread, other windows will not get any messages
- This is what happens under Windows, and causes display of the hourglass mouse pointer
- Under OS/2 programmers are advised that if processing a message will take more than 1/10th second, they should do the processing in a second thread
- Compare Pagemaker under Windows and OS/2 for a dramatic illustration

How to Create a Thread (OS/2 1.x)

```
BYTE abStack[4096];
TID tidThread;

VOID main() {
        DosCreateThread(ThreadFunc, &tidThread, abStack
 + sizeof(abStack));
        .
        .
}

VOID FAR ThreadFunc(VOID)
{
        VioWrtTTY("Message from new thread\n\n", 25, 0);
}
```

How to Create a Thread (OS/2 2.x)

- TID tidThread;
- struct _threadarg{ . . .} threadarg;
- ULONG ThreadFlags;

- VOID main() {
- .
- DosCreateThread(&tidThread, ThreadFunc, &threadarg, ulThreadFlags, STACKSIZE);
- .
- }

- VOID ThreadFunc(VOID *)
- {
- WinSetWindowText(hwnd,"Message from new thread");
- }

_beginthread()

- _beginthread(ThreadFunc, pStack, usStackSize, pParms);
- _beginthread is preferable to DosCreateThread because
  - –It performs initialisation necessary to allow calls to other C run-time library functions
  - –It allows a NULL pointer to be passed for the thread stack address, causing the C run-time library to automatically allocate and deallocate the thread stack as necessary
  - –It allows a pointer to a parameter or structure to be passed to the thread function
- _beginthread can only be used if the program is linked with one of the multithreaded libraries
  - –LLIBCMT.LIB, LLIBCDLL.LIB, CDLLOBJS.LIB (16-bit)
  - –Mark project as multithreaded (WF/2)
  - –Include <MT\headers.h>

Other Thread Functions

- DosExit()
- _endthread(
- DosSuspendThread(tid)
  - –Suspends the specified thread
- DosResumeThread(tid)
  - –Restarts the specified thread
- DosWaitThread(tid, WaitOption)
- DosSetPriority(fScope, fPrtyClass, sChange, id)
  - –Sets the priority of the specified thread or process
- DosGetPID
  - –Retrieves the process, thread and parent-process identifiers for the current process
- DosGetInfoBlocks(PTIB, PPIB)
- DosSleep
  - –Suspends execution of the current thread for the specified time interval (Warning: not in PM threads!)

Thread Types

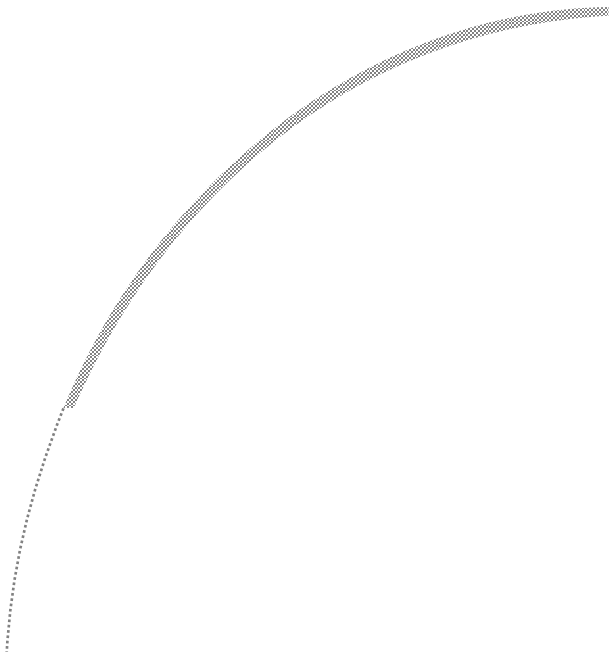
- Message-queue Threads
  - Create message queues and windows
  - Must obey the 1/10th second rule
- Non-message-queue Threads
  - Cannot create windows (since they have no queue to read messages)
  - But can use (e.g.) WinBeginPaint to paint a window belonging to another thread (NB Presentation Spaces are serially reusable)
  - Cannot call WinSendMsg
  - But can call WinPostMsg (typically to signal completion of a task)
  - But are free to take as long as they need to perform tasks (cannot hold up the message queue)

Interprocess Communications

- Anonymous Pipes
- Shared Memory
  - –Giveaway Shared Memory
  - –Named Shared memory
- Queues
- Semaphores
  - –System Semaphores
  - –RAM Semaphores
  - –Fast-safe Semaphores
- Signal Exceptions
- Presentation Manager Facilities
  - –Clipboard
  - –Dynamic Data Exchange

Anonymous Pipes

- Are created from the command line. For example:
  - –DIR | SORT | MORE
- Can be created by a parent process redirecting handles of its children



Named Pipes

- Are actually embedded in the OS/2 kernel
- Allow pipes to be named and extended across the network
- Local/remote operation is transparent
- Pipes can be inbound, outbound or full duplex (virtual circuit abstraction)
- Access to named pipes is subject to user logon permission
- Can be serially reused by different clients
- Can have multiple instances of the same name (e.g. DBMS server pipe pool)
- DOS 3.x and later can access named pipes through the MS-DOS LAN Manager Enhanced redirector

Named Pipe Programming

- At the server, \\NETPC:

```
DosCreateNPipe("\\PIPE\\DBMS");
while(more) {
  DosConnectNPipe();          /* await client)
  DosRead(req);               /* read request */
        .                     /* process request */
  DosWrite(resp);             /* send response */
  DosDisconnectNPipe();       /* close client */
}
```

Named Pipe Programming (cont)

- At the client:

```
DosOpen("\\\\NETPC\\PIPE\\DBMS");
DosWrite(req);
DosRead(resp);
DosClose();
```

or

```
DosOpen("\\\\NETPC\\PIPE\\DBMS");
DosTransactNPipe(req,resp);
DosClose();
```

or

```
DosCallNPipe("\\\\NETPC\\PIPE\\DBMS",req,resp);
```

Named Pipe Programming (cont)

- Open Mode - duplex, inheritance, write-through
- Pipe Mode - Blocking, byte or message mode, instance count
- Buffers can be up to 64 KB in size

```
PSZ     PipeName;
PHPIPE  PipeHandle;
ULONG   OpenMode;
ULONG   PipeMode;
ULONG   OutBufSize;
ULONG   InBufSize;
ULONG   TimeOut;
APIRET  rc;                     /*  Return Code. */
rc = DosCreateNPipe (PipeName,  PipeHandle,
       OpenMode,  PipeMode,  OutBufSize,
       InBufSize,  TimeOut);
```

Named Pipe Functions

- DosCreateNPipe()
- DosConnectNPipe()
- DosDisConnectNPipe()
- DosTransactNPipe()
- DosCallNPipe()
- DosPeekNPipe()
- DosQueryNPHState()
- DosQueryNPipeInfo()
- DosQueryNPipeSemState()
- DosSetNPHState()
- DosSetNPipeSem()
- DosWaitNPipe()

Giveaway Shared Memory (1.x)

- Give-away shared memory
  - –Created with DosAllocSeg with either the SEG_GETTABLE or SEG_GIVEABLE attributes
    - ►*DosAllocSeg(size, &sel, SEG_GETTABLE);*
    - ►*Pass selector to other process*
    - ►*Other process calls DosGetSeg() to validate selector*
  - –or
    - ►*DosAllocSeg(size &sel, SEG_GIVEABLE*
    - ►*DosGiveSeg(sel, pidTarget, &pselTarget);*
    - ►*then pass pselTarget to Target process, which need not call DosGetSeg() to validate before using*
  - –Causes allocation of selectors in the disjoint LDT space so that every process can use the same selector value to access this segment

Giveaway Shared Memory (2.x)

- Give-away shared memory
  - –Created with DosAllocSharedMem with either the OBJ_GETTABLE or OBJ_GIVEABLE attributes
    - ►*DosAllocSharedMem(&pv, NULL, size, OBJ_GETTABLE);*
    - ►*Pass pointer to other process*
    - ►*Other process calls DosGetSharedMem() to validate memory object address*
  - –or
    - ►*DosAllocSharedMem(&pv, NULL, size, OBJ_GIVEABLE);*
    - ►*DosGiveSharedMem(pv, pidTarget, flags);*
    - ►*then pass pv to Target process, which need not call DosGetSeg() to validate before using*

Named Shared Memory

- Exists in the file-system namespace, as \SHAREMEM\name
- Is allocated with DosAllocShrSeg() (1.x)
- or DosAllocSharedMem() (2.x)
- Recipient accesses with DosGetShrSeg() (1.x)
- or DosGetNamedSharedMem() (2.x)
- Remember, what two consenting adult processes do in shared memory is entirely their business! Try to use standard formats (metafiles, standard structures, bitmaps, etc)

Queues

- Can be thought of as 'structured pipes'
- Exist in the file system namespace as '\queues\name'
- Can be written to by any process
- Can only be read by the queue creator
- Can be read in FIFO, LIFO or priority sequence, or peeked in arbitrary sequence
- Offer high performance
- 3192 items max
- Effectively a linked list
- Warning! In OS/2 2.0, both 16-bit and 32-bit versions are provided and have the same function names, but ARE NOT COMPATIBLE. I.e., 16-bit queues cannot be used by 32-bit apps and vice versa

Queue Functions

- DosCreateQueue
- DosOpenQueue
- DosReadQueue
- DosWriteQueue
- DosPurgeQueue
- DosCloseQueue
- DosQueryQueue
- DosPeekQueue

Semaphores in OS/2 1.x

- System semaphores
  - –Used for inter-process communications
  - –Exist in the file system namespace as '\sem\name'
- RAM semaphores
  - –Used for intra-process communications (between threads)
  - –Are simply long integers in the address space of the process (treat with care!)
- Fast-Safe Semaphores
  - –Combine the speed of RAM semaphores with the safety of system semaphores

Semaphore Functions (OS/2 1.x)

- DosCreateSem
  - –Creates a system semaphore
- DosOpenSem
  - –Opens a system semaphore
- DosSemSet
  - –Sets a system or RAM semaphore
- DosSemClear
  - –Clears a system or RAM semaphore
- DosSemRequest
  - –Sets a system or RAM semaphore, if the semaphore is cleared
- DosSemSetWait
  - –Sets a semaphore and waits for it to be cleared
- DosSemWait
  - –Waits for a semaphore to be cleared

OS/2 2.1 Semaphores

- OS/2 2.1 supports three types of semaphores:
  - –Event semaphores
    - ►*signalling mechanism*
  - –Mutex semaphores
    - ►*used to protect access to a critical region, e.g. file update)*
  - –Muxwait semaphores
    - ►*compound semaphore: consists of up to 64 event or mutex semaphores.*
- in two classes
  - –Private
    - ►*up to 64K, for intra-process communications*
  - –Shared
    - ►*up to 64K in system, for inter-process communications*
- and they can be named (\SEM32\name) or unnamed

Event Semaphores

- Any Thread:
- DosCreateEventSem(nam);
- DosResetEventSem(nam);
- .
- .
- .
- DosPostEventSem(nam);

- Any Thread:
- .
- DosOpenEventSem(nam);
- DosWaitEventSem(nam);
- .

- Any Thread:
- .
- DosOpenEventSem(nam);
- DosWaitEventSem(nam);

- Any Thread:
- .
- DosOpenEventSem(nam);
- DosWaitEventSem(nam);

Event Semaphore Functions

- DosCreateEventSem()
- DosOpenEventSem()
- DosCloseEventSem()
- DosPostEventSem()
- DosResetEventSem()
- DosQueryEventSem()
  - Returns the post count of the event sem
- DosWaitEventSem()

Mutex Semaphores

- Any Thread:
- .
- DosCreateMutexSem(nam);
- DosRequestMutexSem(nam);
- .
- .
- DosReleaseMutexSem(nam);

- Any Thread:
- .
- DosOpenMutexSem(nam);
- DosRequestMutexSem(nam);
- .
- DosReleaseMutexSem(nam);

- Any Thread:
- .
- DosOpenMutexSem(nam);
- DosRequestMutexSem(nam);
- .
- DosReleaseMutexSem(nam);

Mutex Semaphore Functions

- DosCreateMutexSem()
- DosOpenMutexSem()
- DosCloseMutexSem()
- DosQueryMutexSem()
- DosRequestMutexSem()
- DosReleaseMutexSem()

Muxwait Semaphore Functions

- DosCreateMuxWaitSem()
- DosOpenMuxWaitSem()
- DosCloseMuxWaitSem()
- DosAddMuxWaitSem()
- DosDeleteMuxWaitSem()
- DosQueryMuxWaitSem()
- DosWaitMuxWaitSem()

Signals

- Are OS/2's primary form of asynchronous communications
- Signals cannot be ignored - they are a form of logical interrupt
- Used by full-screen processes
- Processes should nominate signal-handler functions to deal with (for example) SIG_KILLPROCESS and SIG_CTRLBREAK
- Useful functions:
  - –DosSetSigHandler(Sig_H_Func, &prev_func, &fAction, SIGA_ACCEPT, SIG_CTRLC);
- The signal handler should be prototyped:
  - –void _far _pascal MySigHandler(USHORT usSigArg, USHORT usSigNum);
  - –where usSigNum is the signal type (e.g. SIG_KILLPROCESS)

File Handling

Low Level I/O

Opening a File

- DosOpen() [OS/2 1.0, 1.1 2.X)
  - –USHORT DosOpen(pszFileName, phf, pusAction, ulFileSize, usAttribute, usOpenFlags, usOpenMode, ulReserved)
- DosOpen2() [OS/2 1.2, 1.3]
  - –USHORT DosOpen2(pszFileName, phf, pusAction, ulFileSize, usAttribute, usOpenFlags, ulOpenMode, peaop, ulReserved)

Actions and Attributes

- Actions
  - FILE_CREATED — File was created.
  - FILE_EXISTED — File already existed.
  - FILE_TRUNCATED — The file existed and was truncated to the specified size.
- Attributes
  - FILE_READ_ONLY — File can be read but not written.
  - FILE_HIDDEN — File is hidden and does not appear in directory listings.
  - FILE_SYSTEM — File is a system file.
  - FILE_ARCHIVED — File has been archived.

Open Flags

- Used to specify error handling, e.g. what if creating a file and it already exists?
  - FILE_CREATE Create a new file; fail if it already exists.
  - FILE_OPEN Open an existing file, fail if it does not exist.
  - FILE_OPEN | FILE_CREATE Open an existing file or create a new one
  - FILE_TRUNCATE Open an existing file and change its size.
  - FILE_TRUNCATE | FILE_CREATE Open an existing file and change its size or create a new file of that size.

Open Mode

- OPEN_FLAGS_DASD Opens a physical drive for direct access.
- OPEN_FLAGS_FAIL_ON_ERROR * Bypasses the system critical-error handler.
- OPEN_FLAGS_NOINHERIT * The file handle is not available to any children.
- OPEN_FLAGS_WRITE_THROUGH * system will
- write data to the device before returning.
- OPEN_FLAGS_NO_LOCALITY
- OPEN_FLAGS_SEQUENTIAL The file is accessed sequentially.
- OPEN_FLAGS_RANDOM The file is accessed randomly.
- OPEN_FLAGS_RANDOMSEQUENTIAL The file is accessed randomly, but with a degree of sequential access.
- OPEN_FLAGS_NO_CACHE The disk driver should not cache data in I/O operations on this file.

Access Mode

- OPEN_ACCESS_READONLY — Program can only read from file, not write.
- OPEN_ACCESS_READWRITE — Program can read and write the file.
- OPEN_ACCESS_WRITEONLY — Program can write to the file, but not read.

Share Mode

- OPEN_SHARE_DENYNONE Other processes can open the file for any access mode (read-only, write-only or read-write)
- OPEN_SHARE_DENYREAD Other processes can open the file for write-only access but they cannot open if for read-only or read-write access.
- OPEN_SHARE_DENYREADWRITE The current process has exclusive access to the file. The file cannot be opened by any process.
- OPEN_SHARE_DENYWRITE Other processes can open the file for read-only access but they cannot open it for write-only or read-write access.

Reading and Writing

- USHORT DosRead(hf, pvBuf, cbBuf, pcbBytesRead)
- HFILE hf; /* File to read */
- PVOID pvBuf; /* address of buffer */
- USHORT cbBuf; /* count of bytes in (size of) buffer */
- PUSHORT pcbBytesRead; /* count of bytes actually read */

- USHORT DosWrite(hf, pvBuf, cbBuf, pcbBytesWritten)
- HFILE hf; /* File to write */
- PVOID pvBuf; /* address of buffer */
- USHORT cbBuf; /* count of bytes in (size of) buffer */
- PUSHORT pcbBytesWritten; /* count of bytes actually written */

Moving the File Read/Write Pointer

- USHORT DosChgFilePtr(hf, lDistance, fMethod, pulNewPtr)
- HFILE hf;
- LONG lDistance;
- USHORT fMethod;
- PULONG pulNewPtr;

- The parameter fMethod will be one of the following values:
  - –FILE_BEGIN Start move at the beginning of the file
  - –FILE_CURRENT Move relative to the current position
  - –FILE_END Move relative to the end of the file.
- The parameter lDistance is signed. Positive values move forward, negative values backward through the file.

File and Region Locking

- USHORT DosFileLocks(hf, pfUnlock, pfLock)
- HFILE hf;
- PFILELOCK pfUnlock;
- PFILELOCK pfLock;

- A FILELOCK structure looks like this:
- typedef struct _FILELOCK {
- LONG lOffset;
- LONG lRange;
- } FILELOCK;

Miscellaneous Functions

- DosResetBuffer()
- DosSetMaxFH()
- DosSetFHState()
- DosQueryFHState()
- DosDupHandle()

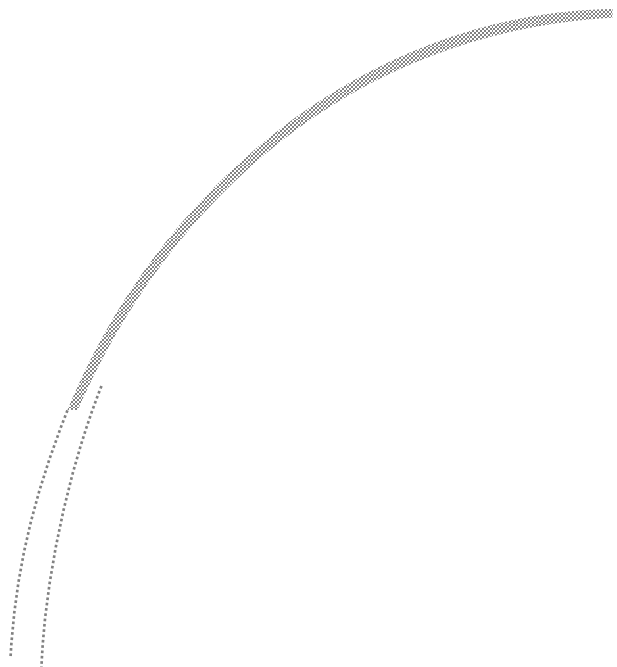
Day 3 – Session 2

Lab Exercise 6 - Threads

Day 3 – Session 3

Workshop

Day 3 – Session 4



Filesystems & EA's

OS/2 File Systems

FAT and HPFS

OS/2 File Systems

- OS/2 originally shipped with the DOS file system as a stop-gap measure
- This was not optimised
- OS/2 1.2 and later support Installable File Systems
- Three major IFS's exist:
  - –High Performance File System
  - –LAN Manager 2.x
  - –CD-ROM IFS

FAT File System

- File Allocation Table and root directory on outer cylinders
- Extensive head movement
- Linear directory searching
- Inefficient allocation in terms of clusters
- Fragmentation of files

High Performance File System

- Directories scattered across disk
- Allocation recorded by bitmaps located in centre of 16 MB bands, close to the files they control
- Directories are B+trees
- Allocation in sectors, not clusters - less wasted space
- Not nearly as susceptible to fragmentation
- Multi-threaded
- Lazy writes on cacheing dramatically improve performance
- Must explicitly shut down to flush cache (though Ctrl-Alt-Del is captured)
- Benefits from large caches (up to 2 MB - up to 60% of machine RAM for HPFS386)

HPFS Features

- High Performance
- Long file names
  - Up to 254 characters long
  - Mixed upper and lower case
  - Can include spaces and other symbols
- Extended Attributes
  - File type
  - Time stamps - creation, update, access
  - Subject
  - Keywords
  - Icon
  - ISO ASN.1

File System Functions

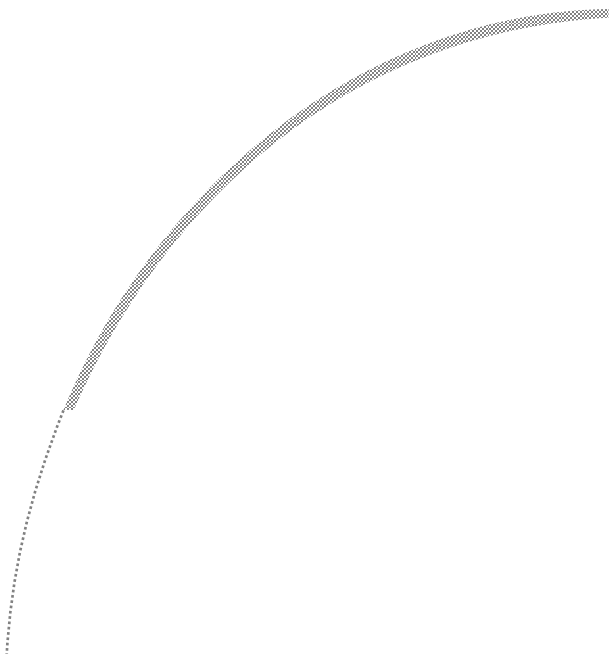
- DosQueryFHState
  - –Queries whether a handle is for a file, pipe or device
- DosFindFirst
- DosFindNext
  - –Search directories for matching files
- DosOpen
- DosClose
- DosRead
- DosWrite

EA Functions

- EA's can be
  - –text
  - –bitmaps
  - –binary
  - –ISO ASN.1
  - –Single-valued
  - –Multi-valued
- Full EA Structure (FEA2) - name and value
- FEA2List - length, then list of FEA2 structures
- Get EA Structure (GEA2) - EA name
- GEA2List - length, then list of GEA2 structures
- EAOP2 Structure - GEA2List, FEA2List and error field

EA Functions (cont)

- DosOpen()
- DosFindFirst
- DosQueryFileInfo() - Level 3
- DosQueryPathInfo() - Level 3
- DosSetFileInfo
- DosSetPathInfo



EA Types

- Stored in first word of EA
- EAT_BINARY
- EAT_ASCII
- EAT_BITMAP
- EAT_METAFILE
- EAT_ICON
- EAT_EA - ASCII Name of another EA
- EAT_MVMT - Multi valued, multi-types
- EAT_MVST - Multi-valued, single-type
- EAT_ASN1 - ISO ASN.1

Standard EA's

- .ASSOCTABLE
- .CODEPAGE
- .COMMENTS
- .HISTORY
- .ICON
- .KEYPHRASES
- .LONGNAME
- .SUBJECT
- .TYPE
- .VERSION

Day 3 – Session 4

Lab Exercise 8 – Directory Listing

Agenda

- Day 1
  - Session 1 – Introduction to Tools
  - Session 2 – Introduction to PM
  - Session 3 – Lab Exercise 1
  - Session 4 – Windows Parentage and Ownership
- Day 2
  - Session 1 – Window Controls
  - Session 2 – Lab Exercise 2 – Menus and Messages
  - Session 3 – Memory Management
  - Session 3 – Lab Exercise 4 – Memory Management
  - Session 4 – Dynamic Link Libraries
  - Session 4 – Lab Exercise 5 – Dynamic Link Libraries

Agenda

- Day 3
  - Session 1 – Threads, IPC and File I/O
  - Session 2 – Lab Exercise 6 - Threads
  - Session 3 - Workshop
  - Session 4 – Filesystems % EA's
  - Session 4 – Lab Exercise 8 – Directory Listing
- Day 4
  - Session 1 – Window Words, Subclassing, Dialogs
  - Session 2 – Lab Exercise 9 – Multiple Windows and Instance Data
  - Session 3 – Lab Exercise 9 continues
  - Session 4 – Standard Dialogs and INI files
- Day 5
  - Session 1 – Graphics Programming Interfase
  - Session 2 - Workshop
  - Session 3 – SOM and WPS
  - Session 4 – It's Friday…

Day 4 – Session 1

Window Words, Subclassing, Dialogs

Advanced PM Programming

Window Words

The PM API

- API Prefixes
  - Dev
  - Dos
  - Drg
  - Gpi
  - Prf
  - Spl
  - Win
- API Verbs
  - Create
  - Destroy
  - Change
  - Set
  - Query
  - Give
  - Get
  - Draw
  - Map
  - Open
  - Close
  - Add
  - Delete
- Nouns
  - Window
  - Text
  - Ptr
  - UShort
  - ULong
  - Atom
  - String
  - Bitmap
  - Pointer
  - Dlg
  - Pos

Instance Data

- Window procedures implement window classes
- Each window is an object of the specified class
- Therefore, multiple windows may share one winproc
- Therefore, winprocs must be reentrant:
  - Automatic variables, OK
  - Static variables, No Way!
- This is *absolutely fundamental* to PM programming!

Allocating Window Words

- The last argument to the WinRegisterClass() function call is the number of bytes of window words to reserve in each window of that class
- So:
  - WinRegisterClass(hab,
  - WC_MYCLASS,
  - (PFNWP)MyWndProc,
  - flClassStyles,
  - sizeof(void *));
- would reserve sufficient space for a pointer.

In the Winproc

- case WM_CREATE:
  - Allocate memory for a structure to hold the window's static variables
  - Then store the pointer or selector into the window with
    - *WinSetWindowPtr(hwnd, 0, p);*
    - *WinSetWindowUShort(hwnd, QWS_USER, sel);*
- At the top of the window procedure, or in individual message stubs, retrieve the pointer or selector with
  - p = WinQueryWindowPtr(hwnd, 0);
  - sel = WinQueryWindowUShort(hwnd, QWS_USER);
- Use a macro for ease and reliability
- Refer to all variables indirectly, using the -> operator
- If any parameters need to be passed to the window at creation, this can be done in a CREATESTRUCT.

Object Windows

- Most windows have an appearance on the screen
- However, windows provide an excellent means of encapsulating data (in window words) and functions (in winprocs)
- This is object-oriented programming, where a window is an object and the winproc is a set of methods for that object
- We can take advantage of this 'feature' of OS/2 by creating 'object windows' which have no on-screen appearance
- To create an object window, call WinCreateWindow with a parent parameter of HWND_OBJECT
- Can be used to encapsulate
  - Network sessions
  - Databases
  - File structures
- Threads not subject to 1/10th second rule

Subclassing

- A powerful object-oriented concept is inheritance.
- We can implement this in OS/2 by subclassing existing window classes.
- For example, subclassing an entry-field control to perform editing and validation
- Or subclassing a MLE control to create a system editor.
- Write a winproc which deals just with new messages you define or over-rides messages defined in the parent
- In the new winproc, replace all calls to WinDefWindowProc with calls to the old window procedure
- To subclass:
  - –oldwndproc = WinSubclassWindow(hwnd, newwndproc);
- Note because the window is subclassed after creation, WM_CREATE cannot be overridden

Dialogs

- Are child windows used for user interaction, usually in question/answer format, over a brief period of time
- A dialog window is a subclassed frame window which organises a set of controls and provides default high-level processing of tab keys, editing etc.
- Can be created by
  - WinDlgBox()
  - WinLoadDlg()
  - WinCreateDlg()

WinDlgBox()

- Produces a modal dialog window which must be dealt with and dismissed before other application windows can obtain focus
- Creates the dialog window from a template stored in the module resources
- Is processed via a programmer-supplied dialog procedure much like a window procedure, except
  - Creation processing done in WM_INITDLG
  - Default should return WinDefDlgProc()
- Removed by the WinDismissDlg() call
- Return the ID of the button used to dismiss the dialog
- Pass parameters and defaults in a structure which must start with a cbSize field

WinLoadDlg()

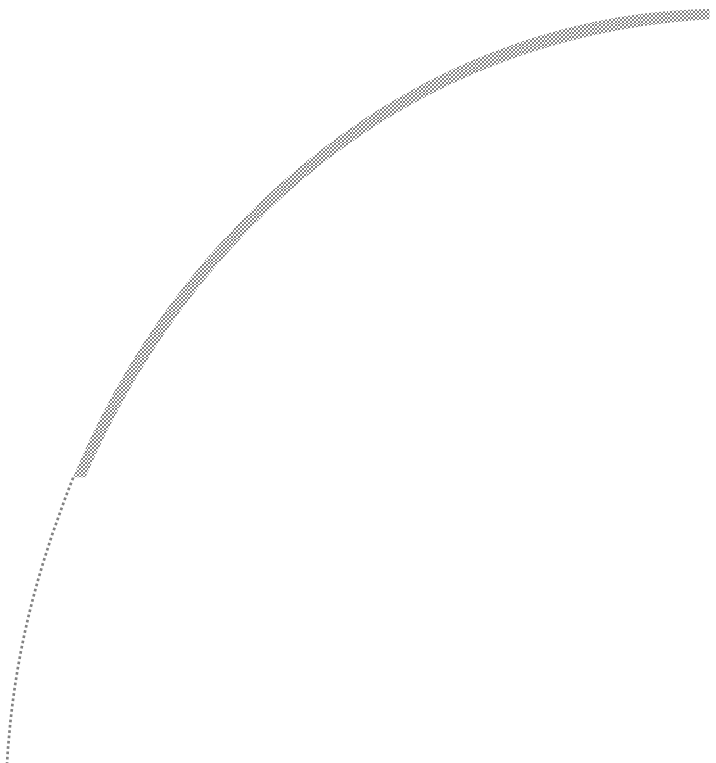
- Produces a modeless dialog box which remains on the screen while the user works with this and other windows (like a standard window)
- Creates the dialog window from a template stored in the module resources
- Is processed via a programmer-supplied dialog procedure much like a window procedure
- Creation processing done in WM_INITDLG
- Can be made modal with the WinProcessDlg() call
- Removed by the WinDismissDlg() call

WinCreateDlg()

- Produces a modeless dialog box which remains on the screen while the user works with this and other windows (like a standard window)
- Creates the dialog window dynamically from a template built in memory
- Is processed via a programmer-supplied dialog procedure much like a window procedure
- Creation processing done in WM_INITDLG
- Can be made modal with the WinProcessDlg() call
- Removed by the WinDismissDlg() call

Dialog Templates

- Can be created by the Dialog Editor (dlgbox.exe)
  - Kindergarten 'paint-by-numbers' exercise
- Or by manual scripting in the resource compiler file
  - hard work!



Day 4 – Session 2

Lab Exercise 9 – Multiple Windows and Instance Data

Day 4 – Session 3

Lab Exercise 9 – …continues

Day 4 – Session 4

Standard Dialogs and INI Files

Standard Dialogs

For Files and Fonts

FILEDLG Structure

```
cbSize;                 /* Size of FILEDLG structure.          */
fl;                     /* FDS_ flags. Alter behavior of dlg. */
ulUser;                 /* User defined field.                 */
lReturn;                /* Result code from dialog dismissal. */
lSRC;                   /* System return code.                 */
pszTitle;               /* String to display in title bar.     */
pszOKButton;                    /* String to display in OK button.     */
pfnDlgProc;                     /* Entry point to custom dialog proc. */
pszIType;               /*initial EA type filter. */
papszITypeList;                 /*  Type strings.  */
pszIDrive;              /* Initial drive.      */
papszIDriveList;        /* Drive strings.  */
hMod;                   /* Custom File Dialog template.        */
szFullFile[CCHMAXPATH];         /* Initial  path and file.       */
papszFQFilename;        /* FQFname*/
ulFQFCount;             /* Number of files selected            */
usDlgId;                /* Custom dialog id.                   */
x;                      /* X coordinate of the dialog          */
sEAType;                /* Selected file's EA Type.            */
```

File Dialog Flags

- FDS_OPEN_DIALOG
- FDS_SAVEAS_DIALOG
- FDS_CENTER
- FDS_CUSTOM
- FDS_HELPBUTTON
- FDS_FILTERUNION
- FDS_PRELOAD_VOLINFO
- FDS_MULTIPLESEL
- FDS_ENABLEFILELB
- FDS_INCLUDE_EAS

File Dialog Messages

- FDM_VALIDATE
- FDM_FILTER
- WM_COMMAND

Using Standard File Dialog

- FILEDLG fildlg;
- HFILE hFile;

- memset(&fildlg, NULL, sizeof(FILDLG));
- fildlg.cbsize=sizeof(FILDLG);
- fildlg.fl = FDS_OPEN_DIALOG | FDS_CENTER | FDS_HELPBUTTON;
- fildlg.pszTitle = "Open Edit File";
- hwndFileDlg = WinFileDlg(HWND_DESKTOP, hwndOwner, &fildlg);
- if(hwndFileDlg && (fildlg.lReturn == DID_OK))
- rc = DosOpen(fild.szFullFile, &hFile, &ulAction, . . .);

Standard File Open/Save Dialog

FONTDLG Structure

cbSize; sizeof(FONTDLG)
hpsScreen; Screen presentation space
hpsPrinter; Printer presentation space
pszTitle; Application supplied title
pszPreview; String to print in preview wndw
pszPtSizeList; Application provided size list
pfnDlgProc; Dialog subclass procedure
pszFamilyname; Family name of font
fxPointSize; Point size the user selected
fl; FNTS_* flags - dialog styles
flFlags; FNTF_* state flags
flType; Font type option bits
flTypeMask; Mask of which font types to use
flStyle; The selected style bits
flStyleMask; Mask of which style bits to use
clrFore; Selected foreground color
clrBack; Selected background color

ulUser Blank field for application
lReturn; Return Value of the Dialog
lSRC; System return code.
lEmHeight; Em height of the current font
lXHeight; X height of the current font
lExternalLeading; External Leading of font
hMod; Module to load custom template
fAttrs; Font attribute structure
sNominalPointSize; Nominal Point Size of font
usWeight; The boldness of the font
usWidth; The width of the font
x; X coordinate of the dialog
y; Y coordinate of the dialog
usDlgId; ID of a custom dialog template
usFamilyBufLen; Length of family buffer provided
usReserved; reserved

Font Dialog Flags

- FNTS_CENTER
- FNTS_CUSTOM
- FNTS_HELPBUTTON
- FNTS_MULTIFONTSELECTION
- FNTS_MODELESS

Font Dialog Messages

- FNTM_FACENAMECHANGED
- FNTM_POINTSIZECHANGED
- FNTM_STYLECHANGED
- FNTM_COLORCHANGED
- FNTM_UPDATEPREVIEW
- FNTM_FILTERLIST
- WM_COMMAND

Font Dialog Standard Controls

- DID_OK_BUTTON
- DID_CANCEL_BUTTON
- DID_FONT_DIALOG
- DID_NAME
- DID_STYLE
- DID_DISPLAY_FILTER
- DID_PRINTER_FILTER
- DID_SIZE
- DID_SAMPLE
- DID_OUTLINE
- DID_UNDERSCORE
- DID_STRIKEOUT
- DID_HELP_BUTTON
- DID_APPLY_BUTTON
- DID_RESET_BUTTON
- DID_NAME_PREFIX
- DID_STYLE_PREFIX
- DID_SAMPLE_GROUPBOX
- DID_EMPHASIS_GROUPBOX

Using the Font Dialog

- FONTDLG fntdlg;
- USHORT usCodePage;
- HPS hpsScreen;

- hpsScreen=WinGetPS(hwnd);
- memset(&fntdlg, NULL, sizeof(FONTDLG));
- fntdlg.cbSize = sizeof(FONTDLG);
- fntdlg.fl = FNTS_CENTER | FNTS_HELPBUTTON;
- fntdlg.pszTitle = "Fonts";
- fntdlg.pszFamilyName = "";
- fntdlg.fxPointSize = MAKEFIXED(12,0);
- fntdlg.usWeight = FWEIGHT_NORMAL;
- fntdlg.usWidth = FWIDTH_NORMAL;

Using the Font Dialog (cont)

- fntdlg.flType = 0L;
- fntdlg.clrFore = CLR_BLACK;
- fntdlg.clrBack = CLR_WHITE;
- fntdlg.fAttrs.usCodePage = usCodePage;
- fntdlg.hpsScreen = hpsScreen;
- hwndFontDlg = WinFontDlg(HWND_DESKTOP, hwndOwner, &fntdlg);
- if(hwndFontDlg && (fntdlg.lReturn == DID_OK))
- WinInvalidateRect(hwnd, NULL, 0L);

INI File Interaction

Storing Persistent Settings

INI Files

- \OS2\OS2.INI
  - Handle: HINI_USERPROFILE
  - Used for application settings
- \OS2\OS2SYS.INI
  - Handle: HINI_SYSTEMPROFILE
  - Used for system settings and objects
- Both files are opened as PMSHELL starts and locked
- The handle HINI_PROFILE reads both initialization files, but writes only to OS2.INI
- Additional application-specific INI files can be opened using the function PrfOpenProfile().

Contents of INI Files

- The files are split into sections, based on application name.
- Each section contains multiple keyname/value combinations
- Can be queried using the function:
- PrfQueryProfileString(HINI_PROFILE,
    - pszAppName,
    - pszKeyName,
    - pszErrorText,
    - pszBuffer,
    - ulbufSize);

- Buffer will contain a list of application names, keynames or values, depending on the passed parameters.

Other Useful Functions

- PrfWriteProfileString()
- PrfWriteProfileData()
- PrfQueryProfileSize()
- PrfCloseProfile()

Agenda

- Day 1
  - Session 1 – Introduction to Tools
  - Session 2 – Introduction to PM
  - Session 3 – Lab Exercise 1
  - Session 4 – Windows Parentage and Ownership
- Day 2
  - Session 1 – Window Controls
  - Session 2 – Lab Exercise 2 – Menus and Messages
  - Session 3 – Memory Management
  - Session 3 – Lab Exercise 4 – Memory Management
  - Session 4 – Dynamic Link Libraries
  - Session 4 – Lab Exercise 5 – Dynamic Link Libraries

Agenda

- Day 3
  - Session 1 – Threads, IPC and File I/O
  - Session 2 – Lab Exercise 6 - Threads
  - Session 3 - Workshop
  - Session 4 – Filesystems % EA's
  - Session 4 – Lab Exercise 8 – Directory Listing
- Day 4
  - Session 1 – Window Words, Subclassing, Dialogs
  - Session 2 – Lab Exercise 9 – Multiple Windows and Instance Data
  - Session 3 – Lab Exercise 9 continues
  - Session 4 – Standard Dialogs and INI files
- Day 5
  - Session 1 – Graphics Programming Interfase
  - Session 2 - Workshop
  - Session 3 – SOM and WPS
  - Session 4 – It's Friday…

Day 5 – Session 1

Graphical Programming Interface

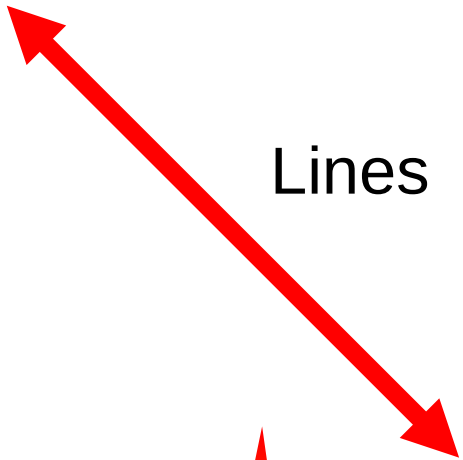
OS/2 Graphics Programming Interface

PM's Graphics 'Language'

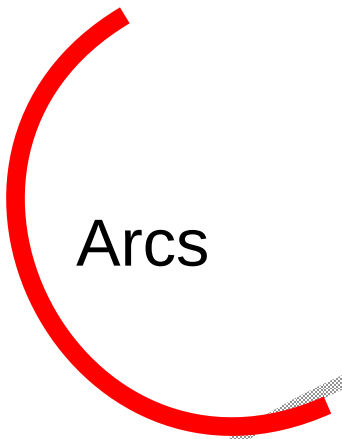
GPI Primitives Can Draw . . .

Text

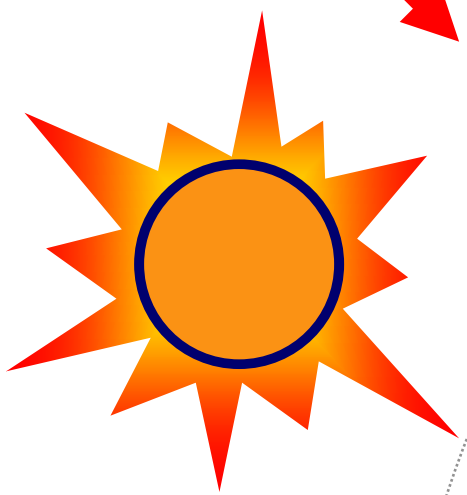
Areas



Lines



Arcs



Bitmaps

GPI Concepts

- The GPI provides all the benefits of
  - a High-Level Graphics Language, which offloads work
- and
  - a Super Device Interface, which decouples devices
- HLGL Features:
  - Stored Picture Elements
  - Hierarchical Picture Construction
  - Picture Editing / Replacement
  - Input / Picture Correlation (Hit Detection)
  - Automatic Picture Repair
- The DI
  - Lets the device do what it can
  - Makes all devices look the same
  - Provides information on the interface

PM's Roots

- GPI's Roots:
  - GDDM (Graphical Data Display Manager)
  - 3270 Graphics Control Program
  - ANSI GKS (Graphical Kernel Standard)
  - PHIGS (Programmer's Hierarchical Interactive Graphics Standard)
- Device Interface:
  - Microsoft Windows GDI
- Terminal Displays
  - Smalltalk
  - dp-CGI (Computer Graphics Interface)

GPI is Different from Earlier Standards

- First system to combine raster and vector support
- Allows device sharing
  - Serial sharing of printers
  - Concurrent use of interactive display

GPI Concepts (cont)

- A Device Context is the means of writing data to an output device. It is both the device driver and the physical device itself (if any).
- Types of device context:
  - Screen device context
  - Memory device context
  - Metafile device context
  - Other device device context (printer, plotter etc)
    - *Information device contexts allows querying*
- Device context is associated with a Presentation Space. Drawing into the PS causes output to the associated DC
- DevEscape function allows direct output.

Opening a Device Context

- Use DevOpenDC() or WinOpenWindowDC()
- Attributes:
  - OD_QUEUED
  - OD_DIRECT
  - OD_INFO
  - OD_METAFILE
  - OD_MEMORY
  - OD_METAFILE_NOQUERY

Presentation Spaces

- A GPI Presentation Space consists of the following:
  - Segment store
  - Definition of symbol sets and fonts
  - Definition of line-type sets
  - Various controls, e.g. draw control
  - Logical color table / color palette
  - Viewing pipeline, down to and including the page and page window
- The Presentation Space is the key to using the GPI
- It is a generalization of the device context of the CPI
- Three types:
  - Normal - full state preserved over time
  - Micro - limited functionality for the expense
  - Micro-cached - usable with one DC only, typically screen

GPI PS Segment Concepts

- A Segment is a collection of picture elements
- Segments may call segments hierarchically
- The set of current segments forms the picture chain
- Segments have many attributes:
  - Detectability, Visibility, Chained, Highlighted, Dynamic
  - Can propagate some attributes to their children
- Segments contain elements (calls to graphics routines)
- Elements are the editable unit
- Edits may be referenced by name or number in insertion or replacement mode
- Elements contain one or more graphics orders
- An order is a byte sequence which represents a drawing primitive

Presentation Space Types

Type	Devices	GPI Calls	VIO Calls	Notes
Cached-micro	Screen	Most	None	Few only Must get and release each time
Micro	All	Most	None	No retained graphics
Normal	All	All	None	Supports all graphics facilities
AVIO (16 Bits)	All	None	Super/Subset	Character-based - row, column

Creating a PS

- Use hps = GpiCreatePS(hab,
- hdc, //Must have device context already
- &sizel, //size of presentation space
- PU_ARBITRARY //Units
- | GPIT_NORMAL //PS type
- | GPIA_ASSOC); //Associate with dc

Presentation Page Units

- Pels
  - PU_PELS
    - *Screen or window coordinates*
    - *Device-dependent*
    - *Aspect ratio may vary*
- Metrics
  - PU_LOMETRIC 0.1 mm
  - PU_HIMETRIC 0.01 mm
  - PU_LOENGLISH 0.01 in
  - PU_HIENGLISH 0.0001 in
  - PU_TWIPS 1/1440 in
    - *Guaranteed sizes for printers and plotters, but not displays*
- Arbitrary
  - PU_ARBITRARY
    - *No measurement scheme, preserves aspect ratio*

GPI Function Groups

- Contexts and Spaces
- Segments
- Transforms
- Clip Shapes
- Colours
- Markers
- Lines
- Arcs
- Fillets and Splines
- Areas
- Patterns
- Images and Bitmaps
- Text
- Regions
- Metafiles

GPI Drawing Primitives

- Lines
  - GpiLine
  - GpiPolyLine
  - GpiBox
- Arcs
  - GpiFullArc
  - GpiPartialArc
  - GpiPointArc
  - GpiPolyFillet
  - GpiPolyFilletSharp
  - GpiPolySpline
- Markers
  - GpiMarker
  - GpiPolyMarker
- Areas
  - GpiBeginArea
  - GpiEndArea
  - GpiBeginPath
  - GpiEndPath
  - GpiFillPath
  - GpiStrokePath
  - GpiOutlinePath
  - GpiModifyPath
  - GpiCloseFigure
- Images
  - GpiImage
  - GpiLoadBitmap
  - GpiCreateBitMap
  - GpiDrawBits
  - WinDrawBitmap
  - GpiBitBlt
  - GpiWCBitBlt

Attributes on Primitives

- Attributes of the various primitives are specified using structures called bundles.
  - Area primitives AREABUNDLE
  - Character primitives CHARBUNDLE
  - Image primitives IMAGEBUNDLE
  - Line primitives LINEBUNDLE
  - Marker primitives MARKERBUNDLE
- The attributes are applied using GpiSetAttrs():
  - LINEBUNDLE linebundle;
  - linebundle.lColor = CLR_RED;

  - GpiSetAttrs(hps, // Handle to PS
  - PRIM_LINE, // type of primitive
  - LBB_COLOR | LBB_WIDTH, // attrs to change
  - LBB_WIDTH, // set to default
  - &linebundle);

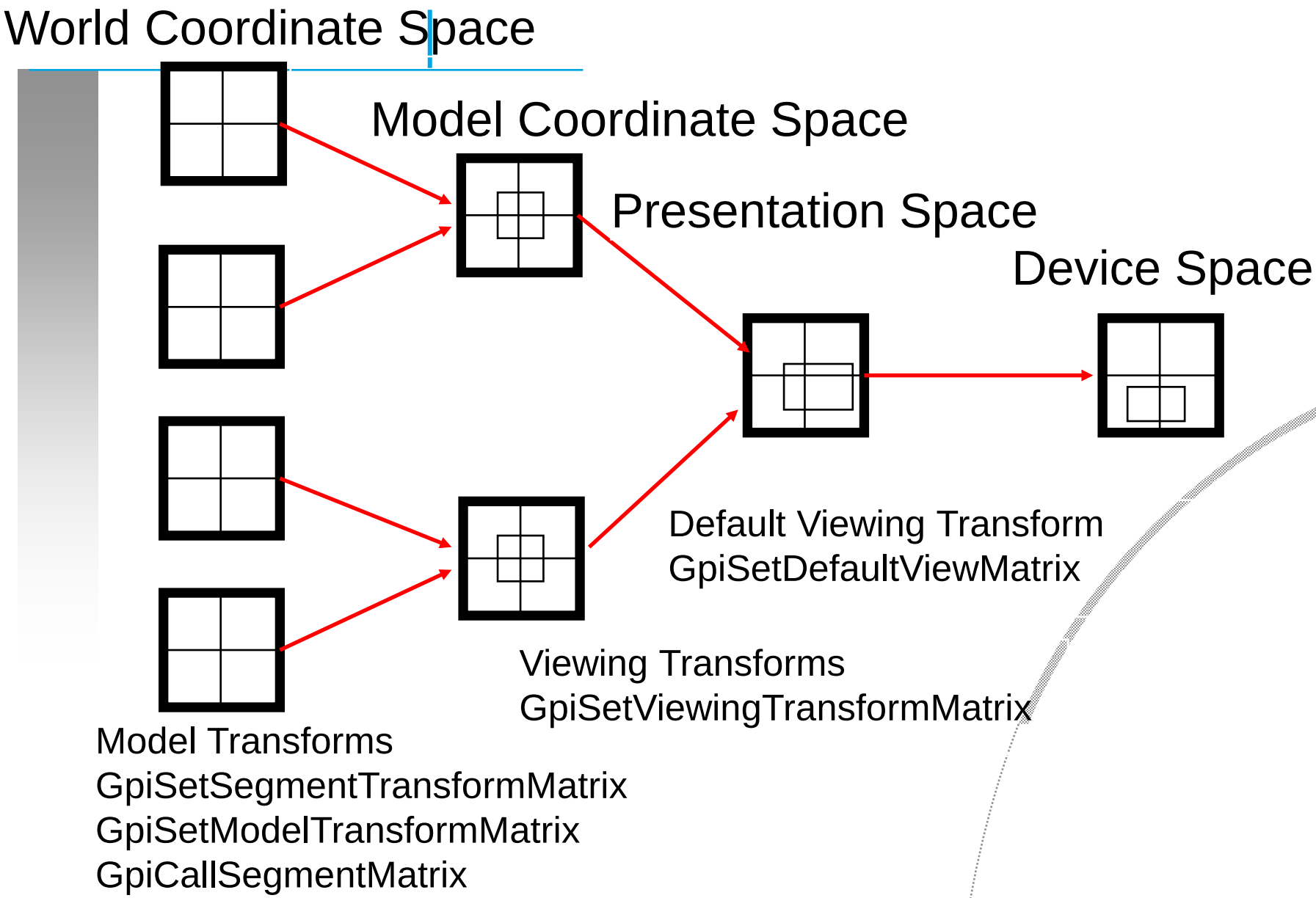
Altering Single Attributes

- Lines
  - GpiSetLineWidth
  - GpiSetLineWidthGeom
  - GpiSetLineType
  - GpiSetLineEnd
  - GpiSetLineJoin
- Areas
  - GpiSetPatternSet
  - GpiSetPattern
  - GpiSetPatternRefPoint
- Markers
  - GpiSetMarkerSet
  - GpiSetMarker
  - GpiSetMarkerBox
- Text
  - –GpiSetCharSet
  - –GpiSetCharMode
  - –GpiSetCharBox
  - –GoiSetCharAngle
  - –GpiSetCharShear
  - –GpiSetCharDirection
- Images
  - –All IMAGEBUNDLE attributes are global
- NB All 'Set' functions have 'Query' equivalents

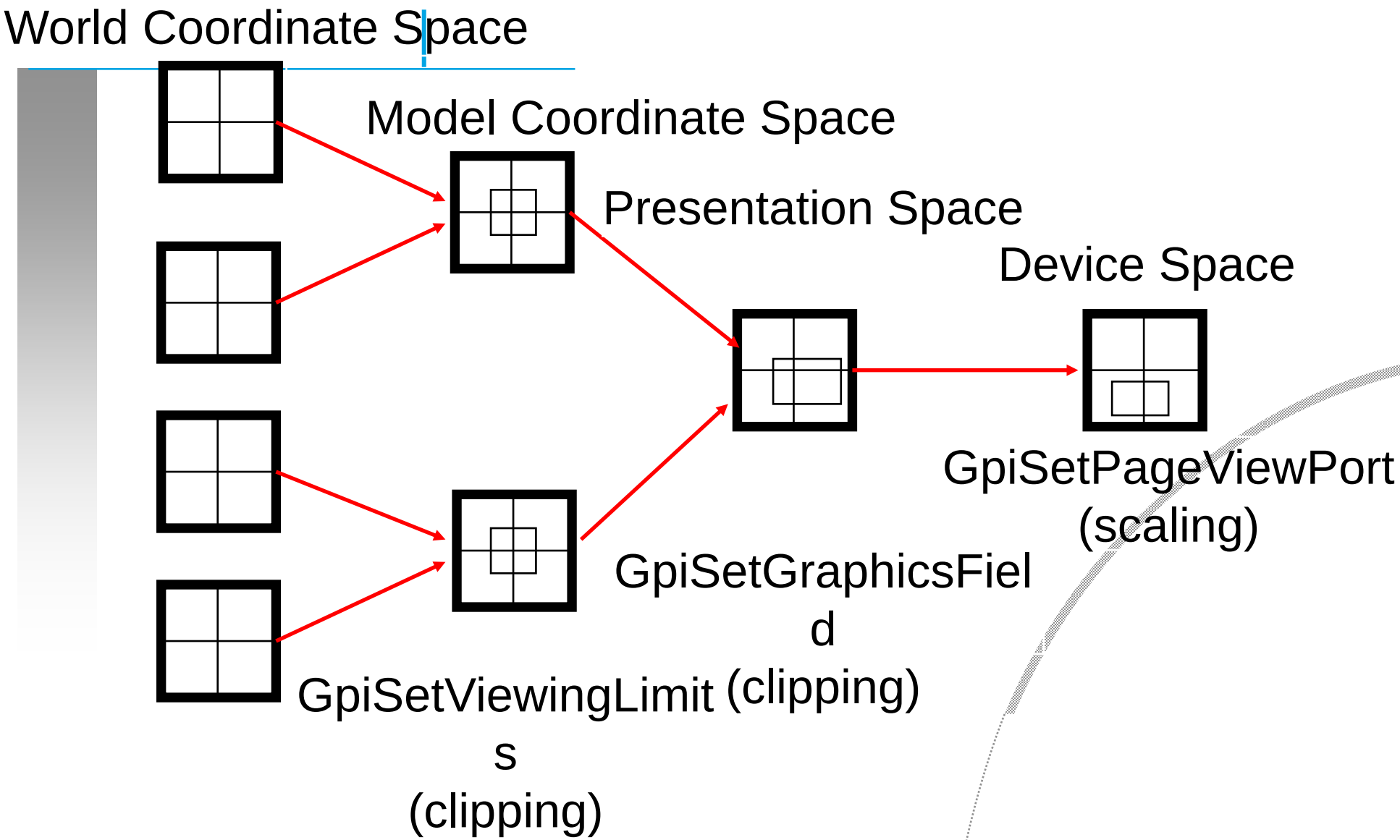
Altering Common Attributes

- Bundled Common Attributes
  - GpiSetColor
  - GpiSetMix
  - GpiSetBackColor
  - GpiSetBackMix
- Non-bundled Common Attributes
  - GpiMove
  - GpiSetCurrentPosition
  - GpiSetArcParams
  - GpiSetViewingLimits
  - GpiSetTag
  - GpiSetModelTransformMatrix
  - GpiSetViewingTransformMatrix

Coordinate Systems and Spaces



Coordinate Systems and Spaces



Day 5 – Session 2

Workshop

Day 5 – Session 3

SOM and WPS

System Object Model

and the Workplace Shell

What is SOM?

- SOM is a system for
  - defining
  - manipulating and
  - releasing
- class libraries
- which is
  - language-independent
  - can be used with both procedural and object-oriented languages
- Three major features:
  - Encapsulation
  - Inheritance
  - Polymorphism

Encapsulation

- An object is implemented as data and methods (public and private) which operate on that data
- Object implementation is hidden (encapsulated) from public view
- SOM permits changes to an object's internal implementation without affecting compatibility:
  - adding new methods
  - adding, changing or deleting unpublished instance variables
  - Inserting new classes above your class in the inheritance hierarchy
  - Relocating methods upward in the class hierarchy
- Implemented as OS/2 DLL's

Inheritance

- The derivation of new child classes from existing parent classes
- Children automatically inherit their parents' methods
- Children can add unique characteristics and new methods
- These children are known as subclasses

Polymorphism

- Is many implementations of the same method for two or more classes of objects.
- Known in SOM as method overrides or override resolution
- SOM supports several forms of polymorphism, in order to support different object-oriented languages

SOM Classes

- SOMObject
  - Basic class with common behaviour for all objects, and no instance data
- SOMClass
  - Base class for all metaclasses (the class of a class is a metaclass). Contains generic constructors
- SOMClassMgr
  - The object that handles class registration and loading from DLL's for each SOM client

Files used in SOM Development

- .CSC - Input file containing the C language-specific class definition (OIDL plus C)
- .H - Output public header file for programs that use this class
- .IH - Output implementation header file
- .PH - Output private header file for private methods
- .C - Output C source template file. Code is added to this to implement object behaviour
- .SC - Ouput OIDL object definition file
- .PSC - Output private definitions of the class interface
- .DEF - link definitions
- .CS2 - Output formatted .CSC file

.CSC File Sections

- Include Section (required)
  - Includes .SC files for parent class, metaclass and ancestors
- Class Section (required)
  - Name, attributes and description of the class
- Release Order Section
  - All new method names and public instance variables
- Metaclass Section
- Parent Class Section
  - Name and description of the parent class
- Passthrough Section
- Data Section
- Methods Section
  - New methods and method overrides

SOM 2

- SOM 2 meets the OMG CORBA spec
- SOM 2 uses IDL rather than OIDL (not much different)
- Benefits:
  - Cross-platform: OS/2, AIX, Windows, Mac, UNIX, AS/400, MVS
  - Industry support: IBM, Apple, Novell/Word Perfect, HP, Sun
  - Cross-language: C, C++, Smalltalk, REXX, etc.
- Supports extensions
  - DSOM - Distributed SOM
    - *'proxy objects' (stubs) allow objects in other processes*
    - *with Workgroup Enabler, allows objects across the network*
  - RSOM - Replicated SOM
  - PSOM - Persistent SOM

Workplace Shell Architecture

- The Workplace Shell is implemented using SOM 1 (OS/2 2.1) or SOM 2 (Warp)
- Base class: wpObject is derived from SOMObject
- From wpsObject, three major subclasses are derived
  - –wpFileSystem
    - ►*persistent data stored in the file system as EA's*
    - ►*e.g. wpFolder, wpDataFile*
  - –wpAbstract
    - ►*persistent data stored in OS2.INI*
    - ►*e.g. wpPalette, wpProgram (program reference), etc*
  - –wpTransient
    - ►*no persistent data*
    - ►*e.g. wpJob*

Designing a WPS App

- Pick the WPS classes that most closely implement your desired behaviour
- Subclass and override/add methods to implement desired behaviour
- Partition logic into client-server to protect both WPSand your app against errant objects (yours or others)
- Debugging: IPMD PMSHELL.EXE and/or learn to use the kernel debugger

Day 5 – Session 4

It’s Friday…

References

- Object-Oriented Programming Using SOM and DSOM
  - Lau, Christina, Van Nostrand Reinhold, 1994, ISBN 0-442-01948-3
- SOMobjects Developer Toolkit Users Guide
  - IBM Part Number 59G5464
- Object-Oriented Design with Applications
  - Booch, Grady, Benjamin/Cummings, 1991, ISBN 0-8053-0091-0